VMClock: Efficient time synchronisation for virtual machines.

v1.0 David Woodhouse <dwmw@amazon.co.uk> 2025-11-26

Background

The requirements for accurate synchronisation of application clocks against real wallclock time are becoming ever more demanding. Increasingly cloud providers are exposing precision clock devices to virtual machines to allow the guest operating systems to synchronise their clocks.

Time on modern systems is typically derived from a CPU-internal counter *(TSC, timebase, arch counter)* which runs at a nominally constant frequency of typically between 1GHz and 4GHz. In practice, the frequency of the underlying hardware counter will vary with environmental conditions, with a tolerance of the order of ±50PPM. It is this variance which must constantly be corrected by synchronising against an external clock.

Synchronisation against an external clock typically works by reading the CPU counter, then reading the external clock, and finally reading the CPU counter again — then assuming that the external clock reading was concurrent with a point in time between the two CPU counter readings to give a pair of { CPU counter, real time} values. Successive such readings are used to calibrate the precise rate at which the CPU counter is running, in order to use it for precision timekeeping.
When applied at scale to virtual machines, there are a number of problems with this approach. Firstly, where virtual CPUs are overcommitted across a smaller number of physical CPUs in a host, guests experience “steal time” — time when their vCPU is not actually running. That steal time is unpredictable and can occur in the critical period between one read of the CPU counter and the next, affecting the precision of the estimated reading.

A remedy for this issue is to repeat the reading a number of times, and to use the result where the latency between first and last CPU counter reading is the lowest. Which exacerbates the second problem, that a large number of separate guest operating systems on the same host are now repeating the same work of calibrating the *same* underlying hardware oscillator.

The third major problem of guest-calibrated time is Live Migration, in which a guest is transparently moved from one host to another for maintenance reasons. When this happens, the guest can experience a step change in both the frequency and the value of the CPU counter. The frequency because the migrated guest is now using a different underlying counter, and the value because correctly setting the counter value seen by the guest is dependent on the time synchronisation of each hypervisor host. After a Live Migration, a guest’s clock should be considered inaccurate until it has been resynchronised from scratch. Failure to do so can lead to data corruption, in cases where database coherency depends on accurately timestamped transactions.

The VMclock device

The VMClock device resolves the above issues by allowing the hypervisor to synchronise the hardware clock against external time, and simply present the results to each guest in a shared memory region in the form of a formula for converting the CPU counter into real time. This allows guests to have precision timestamps even immediately after a Live Migration event, and with no need to provide further clock devices to the guest or for guests to spend their own CPU time on calibration.

For guests which do perform their own additional refinement of the clock via NTP or other means, a disruption signal is provided which allows them to discard any such refinement after Live Migration, and start again with the data from the new hypervisor host.

The vmclock_abi structure

The hypervisor provides a structure in shared memory which is readable by the guest, and advertises it via either ACPI or device-tree devices as described later. Where possible, fields and their values are aligned with the definitions in the proposed virtio-rtc standard. As with virtio, all fields are stored in little-endian form.
The fields up to and including time_type are constant and shall not change during the lifetime of the device.

Offset	Field	Description
0x00	`uint32_t magic`	Magic value 0x4b4c4356 (“VCLK”)
0x04	`uint32_t size`	Size of region containing this structure (typically a full page at the granularity at which the hypervisor maps memory to the guest)
0x08	`uint16_t version`	This standard defines version 1
0x0a	`uint8_t counter_id`	The hardware counter used as the basis for clock readings. The values of this field correspond to the `VIRTIO_RTC_COUNTER_xxx` values: • `0x00: VMCLOCK_COUNTER_ARM_VCNT`: The Arm architectural timer (virtual) • `0x01: VMCLOCK_COUNTER_X86_TSC`: The x86 Time Stamp Counter • `0xFF: VMCLOCK_COUNTER_INVALID`: No precision clock is advertised
0x0b	`uint8_t time_type`	Indicates the type of clock exposed through this interface. The values of this field correspond to the `VIRTIO_RTC_CLOCK_xxx` values, except that smearing of clocks is not supported as it is antithetical to precision: • `0x00: VMCLOCK_TIME_UTC` *(Not recommended)* • `0x01: VMCLOCK_TIME_TAI` • `0x02: VMCLOCK_MONOTONIC` For UTC and TAI, the calculation results in a number of seconds since midnight on 1970-01-01. A monotonic clock has no defined epoch. Since UTC has leap seconds and a given numbered second may occur more than once, its use is **NOT RECOMMENDED** in VMClock. Implementations should advertise TAI, with a correct UTC offset.
0x0c	`uint32_t seq_count`	This field is used to provide a sequence-based read/write lock for the non-constant fields which follow. To perform an update, the device will: • Increment this field to an odd value (with the low bit set). • Change other fields as appropriate. • Increment this field again to an even value.
0x10	`uint64_t disruption_marker`	This field is changed each time there may be a disruption to the hardware counter referenced by counter_id, for example through live migration to a new hypervisor host.
0x18	`uint64_t flags`	Feature flags. • Bit 0: `VMCLOCK_FLAG_TAI_OFFSET_VALID` Indicates that the tai_offset field below contains a correct value. • Bit 1: `VMCLOCK_FLAG_DISRUPTION_SOON` Indicates that a clock disruption event (e.g. live migration) is expected to happen in the next day or so. • Bit 2: `VMCLOCK_FLAG_DISRUPTION_IMMINENT` Indicates that a clock disruption event is expected to happen within the next hour or so. • Bit 3: `VMCLOCK_FLAG_PERIOD_ESTERROR_VALID` • Bit 4: `VMCLOCK_FLAG_PERIOD_MAXERROR_VALID` • Bit 5: `VMCLOCK_FLAG_TIME_ESTERROR_VALID` • Bit 6: `VMCLOCK_FLAG_TIME_MAXERROR_VALID` These flags indicate the presence of valid information for the estimated and maximum error fields for the counter period and reference time. • Bit 7: `VMCLOCK_FLAG_VM_GEN_COUNTER_PRESENT` Indicates that the vm_generation_counter field is present. • Bit 8: `VMCLOCK_FLAG_NOTIFICATION_PRESENT` Indicates that the VMClock device will send an interrupt or ACPI notification every time it updates `seq_count` to a new even value. Unknown flags set by the device can safely be ignored. If a change in behaviour is required by a future version of this specification, it would come with a new value of the version field or a new time_type to avoid breaking compatibility with existing users.
0x20	`uint16_t pad`	Unused

0x22	`uint8_t clock_status`	Synchronisation status of the clock: • `0x00: VMCLOCK_STATUS_UNKNOWN:` The clock is in an indeterminate state. Clock parameters in the VMClock structure are not valid and should not be relied upon. • `0x01: VMCLOCK_STATUS_INITIALIZING:` The clock is being initialized and is not yet synchronized. Clock parameters in the VMClock structure are not valid and should not be relied upon. • `0x02: VMCLOCK_STATUS_SYNCHRONIZED:` The clock is synchronized. Clock parameters in the VMClock structure are expected to be correct and may be relied upon. • `0x03: VMCLOCK_STATUS_FREERUNNING:` The clock has transitioned away from being synchronized and is in a free-running state. Clock parameters in the VMClock structure are expected to be valid and may be relied upon. • `0x04: VMCLOCK_STATUS_UNRELIABLE:` The clock is considered broken. Clock parameters in the VMClock structure should not be relied upon.
0x23	`uint8_t leap_second_smearing_hint`	The time exposed through the VMClock device shall never be smeared. This field corresponds to the 'subtype' field in virtio-rtc, which indicates a smearing method. In this case it merely provides a hint to the guest operating system, such that if the guest OS wants to provide its users with an alternative clock which does not follow UTC, it may do so in a fashion consistent with the other systems in the nearby environment. • `0x00: VMCLOCK_SMEARING_STRICT` • `0x01: VMCLOCK_SMEARING_NOON_LINEAR` • `0x02: VMCLOCK_SMEARING_UTC_SLS`
0x24	`int16_t tai_offset_sec`	Signed offset from TAI to UTC at the reference time specified in `time_sec` and `time_frac_sec`, in seconds. Valid if the corresponding bit in the flags field is set. Implementations SHOULD populate this field; the value at time of writing is 37 and is expected to remain so until/unless the next leap second is announced.
0x26	`uint8_t leap_indicator`	Indicates the presence and direction of a leap second occurring in the near future or recent past. Where a VMClock device exposes UTC instead of TAI, this allows the user to differentiate between the first and second occurrences of the second with the same numeric value. Where a guest is smearing time to preserve monotonicity in the vicinity of a leap second, this information allows for that too. *(This field was present in earlier versions of the proposed virtio-rtc specification but was removed.)* This value of this field shall be valid for the point in time referenced by the `time_sec` and `time_frac_sec` fields. • `0x00: VMCLOCK_LEAP_NONE`: No known nearby leap second • `0x01: VMCLOCK_LEAP_PRE_POS`: A positive leap second will occur at the end of the present month • `0x02: VMCLOCK_LEAP_PRE_NEG`: A negative leap second will occur at the end of the present month • `0x03: VMCLOCK_LEAP_POS`: A positive leap second is currently occurring *(set during the 23:59:60 second)* • `0x04: VMCLOCK_LEAP_POST_POS`: A positive leap second occurred at the end of the previous month • `0x05: VMCLOCK_LEAP_POST_NEG`: A negative leap second occurred at the end of the previous month
0x27	`uint8_t counter_period_shift`	Additional shift applied to all the `counter_period*_frac_sec` fixed-point fields.
0x28	`uint64_t counter_value`	Value of the hardware counter at the time represented by `time_sec` + `time_frac_sec`.
0x30	`uint64_t counter_period_frac_sec`	Period of a single counter tick, in units of 1 >> (64 + `counter_period_shift`)
0x38	`uint64_t counter_period_esterror_rate_frac_sec`	Estimated ± error of `counter_period_frac_sec` in the same units.
0x40	`uint64_t counter_period_maxerror_rate_frac_sec`	Maximum ± error of counter_period_frac_sec in the same units.
0x48	`uint64_t time_sec`	Reference time point, seconds since epoch defined by time_type field.
0x50	`uint64_t time_frac_sec`	Fractional part of reference time, in units of second / 2^{64}.
0x58	`uint64_t time_esterror_nanosec`	Estimated ± error of the time given in `time_sec` + `time_frac_sec`, in nanoseconds
0x60	`uint64_t time_maxerror_nanosec`	Maximum ± error of the time given in `time_sec` + `time_frac_sec`, in nanoseconds
0x64	`uint64_t vm_generation_count`	A change in this field indicates that the guest has been loaded from a snapshot. In addition to handling a disruption in time (which will also be signalled through the `disruption_marker` field), a guest may wish to discard UUIDs, reset network connections or reseed entropy, etc.

Notification

After incrementing the `seq_count` field to an even value after changing the non-constant fields in the structure, a VMClock device may raise an interrupt or ACPI notification. VMClock devices that raise interrupts on new `seq_count` values must set the `VMCLOCK_FLAG_NOTIFICATION_PRESENT` bit in `flags` field to advertise the capability.

Calculating time

The VMClock structure provides the following values:

- Reference time T_1 in the `time_sec` and `time_frac_sec` fields
- Counter value C_1 of the hardware counter at time T_1 in the `counter_value` field.
- The period P of a single counter tick is given by `counter_period_frac_sec` >> `counter_period_shift`. For example, a 1GHz clock would have a period of 1ns, which could naïvely be represented as 0x44B82FA0A / 2^{64} by putting that value in `counter_period_frac_sec`. Over long periods of time, however, the loss of precision would be noticeable. So the same 1ns period should be more precisely represented as 0x89705F4136B4A597 / $2^{(64+29)}$ by using that value in `counter_period_frac_sec` and setting `counter_period_shift` to 29.

To calculate the time, the guest shall first read the `seq_count` field and wait until it returns an even value, then read the hardware counter C_{now}. and calculate the time accordingly. Finally, read the `seq_count` field again. If the value of the `seq_count` field has changed, discard the result and repeat the procedure from the beginning.

Where UTC is involved, a correct implementation will need to cope with the case where a leap second has occurred since the reference time T_1, and the result needs to be adjusted accordingly. The `leap_indicator` field exists to resolve the technical ambiguity but using TAI is simpler and less error prone. It is strongly recommended that implementations use TAI as the time standard and advertise a correct TAI offset, to avoid this complexity.

Time error calculation

The VMClock structure optionally advertises maximum error bounds for the clock data it provides. The earliest and latest possible time are calculated from the per-count and overall reported maxerror fields.

The device may update the time calibration fields at any time, by incrementing the `seq_count` to an odd value, adjusting the parameters, then incrementing `seq_count` again to an even value. For any given historical counter reading and the error bounds calculated according to VMClock at that moment, it is guaranteed that any *subsequent* update to the VMClock fields shall also result in a calculation for that same counter value which falls between the earliest and latest times that were previously indicated.

Discovery via ACPI

To expose VMClock to a guest virtual machine, hypervisors need to:

1. Expose a device somewhere in the ACPI namespace with:
    a. a hardware ID (_HID) of “AMZNC10C”
    b. a DOS Device Name ID (_DDN) of “VMCLOCK”
    c. a compatible ID (_CID) of “VMCLOCK”
2. Attach to the device a “_CRS” method which when evaluated describes the shared memory page where the hypervisor has stored the `vmclock_abi` structure. The page needs to reside in an otherwise unused region of guest physical memory.
3. Optionally, the device can (and should) raise an ACPI Notify operation using notification code 0x80, every time the `seq_count` field changes in a new even number. If implemented, the hypervisor must advertise the notification feature to the driver by setting the `VMCLOCK_FLAG_NOTIFICATION_PRESENT` bit in the `flags` field.