The objective of the borrowed virtual time (BVT) algorithm is to support low-latency dispatching of

real-time applications as well as a weighted sharing of the CPU among several classes of applications

[107]. Like SFQ, the BVT algorithm supports scheduling of a mix of applications, some with hard,

some with soft real-time constraints, and applications demanding only a best effort.

Thread i has an effective virtual time, Ei, an actual virtual time, Ai , and a virtual time warp, Wi .

The scheduler thread maintains its own scheduler virtual time (SVT), defined as the minimum actual

virtual time Aj of any thread. The threads are dispatched in the order of their effective virtual time, Ei,

a policy called the earliest virtual time (EVT).

The virtual time warp allows a thread to acquire an earlier effective virtual time – in other words, to

borrow virtual time from its future CPU allocation. The virtual warp time is enabled when the variable

warpBack is set. In this case a latency-sensitive thread gains dispatching preference as

Ei ←

Ai if warpBack = OFF

Ai − Wi if warpBack = ON.

(6.59)

The algorithm measures the time in minimum charging units (mcu) and uses a time quantum called

context switch allowance (C), which measures the real time a thread is allowed to run when competing

with other threads, measured in multiples of mcu. Typical values for the two quantities are

mcu = 100 μsec and C = 100 msec. A thread is charged an integer number of mcu.

Context switches are triggered by traditional events, the running thread is blocked waiting for an

event to occur, the time quantum expires, and an interrupt occurs. Context switching also occurs when a

thread becomes runnable after sleeping. When the thread τi becomes runnable after sleeping, its actual

virtual time is updated as follows:

Ai ← max[Ai , SV T]. (6.60)

This policy prevents a thread sleeping for a long time to claim control of the CPU for a longer period

of time than it deserves.

If there are no interrupts, threads are allowed to run for the same amount of virtual time. Individual

threads have weights; a thread with a larger weight consumes its virtual time more slowly. In practice,

each thread τi maintains a constant ki and uses its weight wi to compute the amount   used to advance

its actual virtual time upon completion of a run:

Ai ← Ai +  . (6.61)

Given two threads a and b,

  = ka

wa

= kb

wb

. (6.62)

The EVT policy requires that every time the actual virtual time is updated, a context switch from the

current running thread τi to a thread τ j occurs if

Aj   Ai − C

wi

. (6.63)

Example 1. The following example illustrates the application of the BVT algorithm for scheduling

two threads a and b of best-effort applications. The first thread has a weight twice that of the second,

wa = 2wb; when ka = 180 and kb = 90, then   = 90.

We consider periods of real-time allocation of C = 9 mcu. The two threads a and b are allowed to

run for 2C/3 = 6 mcu and C/3 = 3 mcu, respectively.

Threads a and b are activated at times

a : 0, 5, 5 + 9 = 14, 14 + 9 = 23, 23 + 9 = 32, 32 + 9 = 41, . . .

b : 2, 2 + 9 = 11, 11 + 9 = 20, 20 + 9 = 29, 29 + 9 = 38, . . .

(6.64)

The context switches occur at real times:

2, 5, 11, 14, 20, 23, 29, 32, 38, 41, . . . (6.65)

The time is expressed in units of mcu. The initial run is a shorter one, consists of only 3 mcu; a context

switch occurs when a, which runs first, exceeds b by 2 mcu.

Table 6.5 shows the effective virtual time of the two threads at the time of each context switch. At

that moment, its actual virtual time is incremented by an amount equal to   if the thread was allowed to run for its time allocation. The scheduler compares the effective virtual time of the threads and first

runs the one with the minimum effective virtual time.

Figure 6.11 displays the effective virtual time and the real time of threads a and b. When a thread

is running, its effective virtual time increases as the real time increases; a running thread appears as a

diagonal line. When a thread is runnable but not running, its effective virtual time is constant. A runnable

period is displayed as a horizontal line.We see that the two threads are allocated equal amounts of virtual

time, but thread a, with a larger weight, consumes its real time more slowly.

Example 2. Next we consider the previous example, but this time there is an additional thread, c, with

real-time constraints. Thread c wakes up at time t = 9 and then periodically at times t = 18, 27, 36, . . .

for 3 units of time.

Table 6.6 summarizes the evolution of the system when the real-time application thread c competes

with the two best-effort threads a and b. Context switches occur now at real times

t = 2, 5, 9, 12, 14, 18, 21, 23, 27, 30, 32, 36, 39, 41, . . . (6.66)

The context switches at times

t = 9, 18, 27, 36, . . . (6.67) are triggered by the waking up of thread c, which preempts the currently running thread. At t = 9 the

time warp Wc = −60 gives priority to thread c. Indeed,

Ec(9) = Ac(9) − Wc = 0 − 60 = −60 (6.68)

compared with Ea(9) = 90 and Eb(9) = 90. The same conditions occur every time the real-time thread

c wakes up. The best-effort application threads have the same effective virtual time when the real-time

application thread finishes and the scheduler chooses b to be dispatched first. Note that the ratio of real

times used by a and b is the same, as wa = 2wb.

Figure 6.12 shows the effective virtual times for the three threads a, b, and c. Every time thread c

wakes up, it preempts the current running thread and is immediately scheduled to run.