

## Reservation-Based Interrupt Scheduling

Nicola Manica, Luca Abeni, Luigi Palopoli

*University of Trento*

*Trento - Italy*

*nicola.manica@disi.unitn.it, luca.abeni@unitn.it, palopoli@dit.unitn.it*

**Abstract**—Some real-time kernels (such as a recent real-time version of Linux) permit to execute interrupt handlers in dedicated threads, to control their interference on real-time applications. However, from the stand-point of real-time analysis, such threads are challenging and cannot be dealt with in the traditional ways. Furthermore, the application of traditional scheduling solutions (such as fixed priorities) proves ineffective in striking a good trade-off between predictability and hardware performance. This paper shows how the problem can be tackled by using the resource reservation abstraction and an appropriate model for schedulability analysis.

### I. INTRODUCTION

A recent trend in real-time research is to support complex applications for which the restrictive assumptions made by the standard real-time theory are not applicable. For example, when real-time applications heavily interact with the external environment (reading or writing large amount of data from hardware devices), the classical model of real-time tasks as independent activities, activated with a given temporal pattern, and using only one type of resources (typically the CPU) is at risk of losing fundamental details. Indeed, I/O operations (and hence the execution of device drivers) can generate timing problems of unexpected harshness when they are not adequately handled and taken into account.

From the scheduling point of view, device drivers can introduce very long priority inversions, potentially disrupting the temporal guarantees. Therefore, a necessary condition to process device information in real-time system is to execute them inside schedulable entities (typically threads). In the past, this solution has been mainly used in  $\mu$ kernel based systems or in small real-time kernels. More recently, even widely used OS kernel such as Linux offered the possibility of executing interrupt handlers in dedicated threads, by applying a real-time patch to the kernel. The recent adoption of interrupt threads in the main stream Linux kernel (starting from version 2.6.30) raises strong expectations on the future popularity of these technique among developers.

In this new context, a very important issue is the impact of the scheduling policy on the device drivers performance. As of today, interrupt threads are frequently scheduled with a fixed priority assigned in empirical ways. The result is often unsatisfactory: the system has to be designed with a conservative assignment of resources and temporal guarantees are difficult to provide. Hence, there is a pressing need for

theoretical models for interrupt threads scheduling enabling well-founded choices of the scheduling parameters.

The scheduling analysis of interrupt threads cannot be carried out using the standard approaches because many hardware devices have finite queues which impose an upper bound on the number of pending interrupts (e.g., interrupts awaiting the CPU allocation). This situation is not normally considered in the standard real-time scheduling theory. Finite queues are obviously dealt with in the framework of queueing theory [1], but this type of analysis is not directly concerned with real-time constraints.

Besides the difficulty in modelling the scheduling problem, the use of standard scheduling solutions could be inadequate in many situation of interest. For instance Modena and other [2] showed that by using fixed priority it is impossible to guarantee *both* the timing constraints of real-time tasks and the performance of hardware devices. In other words, we can easily find situations in which there does not exist a priority assignment allowing us to respect the constraints of real-time tasks and to maintain an acceptable value of throughput for non real-time applications at once. An example of this kind is discussed in Section III-A.

The adoption of advanced scheduling solution such as reservation-based scheduling [3], [4], [5] allows us to overcome this difficulty, as shown in Section V-B. The basic idea of reservation-based interrupt scheduling is that each interrupt thread can be reserved a fraction of the CPU time. This approach (as compared to fixed priorities) enables a finer grained control of the CPU. In this framework, the scheduling design problem becomes identifying the correct choice of the reservation parameters. Standard techniques for dimensioning these parameters [6] require that the inter-arrival time of the threads be large enough compared to the reservation period (which is one of the parameters to choose). However, in the case of interrupt threads, this assumption fails because the inter-arrival time of interrupts can be very small.

To cope with this problem, two novel analysis techniques (the first one deterministic and the second one stochastic) have been developed to identify a correct choice of the reservation parameters so that the number of pending interrupts can be controlled. Finally, an extensive experimental validation carried out on a real implementation is presented that validates the theoretical analysis and proves that the ap-

proach based on interrupt threads and resource reservations is actually viable in real-life situations and it enables a good trade-off between predictability and throughput.

### A. Related Work

The effects of device drivers on user space applications have been previously analysed mainly considering network devices and the network subsystem. Some modifications of such subsystem have been previously proposed in order to improve the network throughput, or to eliminate the so called “receive livelock” [7], [8], [9], but, excluding some notable exceptions [10], [11], real-time aspects have not been investigated.

Interrupt handlers have been traditionally executed inside dedicated threads in  $\mu$ kernel environments [12], [13], and the idea of scheduling device drivers have been previously considered also in real-time theory [14]. Reservation-based scheduling has also been proposed in the past as a way to serve interrupt threads [15]. However, none of the papers mentioned above provide a theoretical analysis of a system including interrupt threads. Moreover, they require heavy modifications to the kernel structure, while the approach proposed in this paper is based on a widely-supported kernel (Linux) plus two fairly maintainable patches [16], [17].

More recently, interrupt threads scheduling in Linux has been evaluated in the context of real-time scheduling [18], [19]. However, the experiments and analysis presented in such works are limited to fixed priorities, and fixed priorities do not seem to provide enough flexibility to properly serve device drivers without affecting real-time tasks [2].

### B. Structure of the Paper

The rest of this paper is organised as follows: Section III provides some definitions and introduces the problem; Section IV presents the theoretical analysis and some simulative validation; Section V describes an implementation of the proposed approach and some experiments on real hardware. Finally, Section VI concludes the paper and describes our future work.

## II. DEFINITIONS AND BACKGROUND

Real-time systems are traditionally modelled as sets  $\Gamma = \{\tau_i, i \in \mathcal{N}\}$  of real-time tasks<sup>1</sup>  $\tau_i$ , where a real-time task is a stream of jobs (or instances)  $J_{i,j}$ . Job  $J_{i,j}$  becomes ready for execution (arrives) at time  $r_{i,j}$ , it requires a computation time  $c_{i,j}$ , and finishes at time  $f_{i,j}$ . Each job  $J_{i,j}$  is also characterised by a deadline  $d_{i,j}$  that is respected if  $f_{i,j} \leq d_{i,j}$ , and is missed if  $f_{i,j} > d_{i,j}$ . Task  $\tau_i$  is often described by a Worst Case Execution Time (WCET)  $C_i = \max_j \{c_{i,j}\}$  and a Minimum Interarrival Time  $P_i = \min_j \{r_{i,j+1} - r_{i,j}\}$ .

In this paper, we will use reservation based scheduling: each task  $\tau_i$  is served by a reservation  $RSV_i = (Q_i^s, T_i^s)$ ,

<sup>1</sup>The word “task” is used to identify a schedulable entity, being it a thread or a process.

meaning that a time  $Q_i^s$  is reserved to  $\tau_i$  in a period  $T_i^s$ ;  $Q_i^s$  is called *maximum budget*, and  $T_i^s$  is called *server period*. Resource reservations present the great advantage of providing *temporal isolation* between tasks, meaning that the worst case behaviour of task  $\tau_i$  is not affected by the other tasks running in the system. As a result, the performance of each task can be analysed in isolation, without considering all the other tasks; this is very important from the analysis point of view, because a system containing IRQ threads can be very complex, and including all the possible interrupts and tasks in the model can make it intractable.

The reservation mechanism used in the paper is the Constant Bandwidth Server (CBS) [4]. When a task  $\tau_i$  is served, the CPU time consumed by it is accounted by decreasing a variable  $q_i$  called *budget*, and tasks are scheduled based on *scheduling deadlines*  $d_i^s$  assigned by the CBS (the client with the earliest scheduling deadline is scheduled). When  $\tau_i$  is created,  $q_i$  and  $d_i^s$  are initialised to 0. When a job  $J_{i,j}$  arrives at time  $r_{i,j}$ ,  $\tau_i$  becomes ready for execution, and the CBS has to assign a scheduling deadline to it. If  $r_{i,j} < d_i^s - \frac{q_i}{Q_i^s} T_i^s$ , then the latest scheduling deadline  $d_i^s$  (and the latest budget  $q_i$ ) can be used; otherwise, a new scheduling deadline  $d_i^s = r_{i,j} + T_i^s$  is generated and the budget  $q_i$  is replenished to  $Q_i^s$ . When  $q_i$  arrives to 0  $\tau_i$  is said to be depleted, and two different behaviours are possible:

- the budget is immediately replenished to  $Q_i^s$  and the scheduling deadline is postponed to  $d_i^s + T_i^s$  (so,  $\tau_i$  remains schedulable).
- $\tau_i$  is not schedulable until time  $d_i^s$ , when the budget will be replenished and the deadline will be postponed as above (so, it cannot be scheduled until  $d_i^s$ ). This is known as *hard reservation behaviour*.

The CBS is guaranteed to execute respecting the real-time properties of the tasks as long as  $\sum_i \frac{Q_i^s}{T_i^s} \leq 1$ ; for a further discussion on the CBS algorithm and on its properties, the reader is referred to the original paper [4].

## III. THE PROBLEM

Based on the system model introduced in Section II, which only considers applications using one hardware resource (the CPU), the schedulability of a real-time task set (i.e., the fact that the tasks’ deadlines are respected) can be guaranteed by using a *proper scheduling algorithm* and an *admission test*. The admission test can be carried out in different ways (utilisation-based test, response time analysis, or time demand analysis) and is traditionally based on  $C_i$  and  $P_i$ . However, this model is simplistic because it does not consider the time consumed by the OS kernel to handle interrupts raised by the hardware devices connected to the system. For example, consider a traditional kernel, in which hardware interrupts are generally served in two phases:

- a short *Interrupt Service Routine* (ISR) is invoked as soon as an interrupt fires and is responsible for ac-

knowledging the hardware interrupt mechanism, postponing the real data transfer and processing to a longer routine, to be executed later;

- a longer routine (*soft interrupt*, or *bottom half*) is executed later to correctly manage the data coming from the hardware device.

ISRs generally execute with interrupts disabled, while soft interrupts always execute with interrupts enabled and are served when switching from kernel space (where ISRs run) to user space (where user programs are executed). Therefore, soft interrupts can be preempted by ISRs.

Both ISRs and soft interrupts have a higher priority than user tasks, and can “steal” execution time from them. Such “stolen time” can be accounted for in real-time guarantees by modelling it as a blocking time  $B_i$  (the admission tests mentioned above can be enhanced to account the blocking times). This implies that a low-priority task can affect the schedulability of high-priority tasks by causing the generation of a large number of hardware interrupts.

This problem is generally solved in real-time kernels by scheduling the interrupt handlers: for example, the Real-Time Preemption patch (RT-preempt) [16] introduces real-time features in the Linux kernel and transforms ISRs and soft interrupts in kernel threads (the hard IRQ thread and the soft IRQ thread), that are *schedulable entities* handled by the kernel scheduler in the same way as user tasks (so, IRQ threads can have lower priorities than real-time tasks, and can be preempted by them). Hence, some new tasks  $\tau_i$  are added to the system, representing the IRQ threads which are used to serve hardware interrupts. The arrival times  $r_{i,j}$  of these new tasks correspond to the times when the interrupts fire.

This solution can present a slightly higher overhead, and requires a more careful synchronisation, but has the advantage to correctly accounting the handler code in a real-time system (that is, the CPU time required to execute the handler can be correctly accounted in order not to break the system’s guarantees).

#### A. A Motivational Example

The possibility to schedule interrupt handlers (provided by IRQ threads) permits to reduce the interference from hardware devices (by giving user-space real-time tasks higher priorities than interrupts). However, identifying a correct scheduling policy for this type of threads is largely an open issue, as discussed in the following example. On a Linux based systems (see Section V for a more detailed description of the experimental setup), we have executed a periodic real-time task  $\tau = (C = 20ms, P = 50ms)$  along with a non real-time task receiving UDP packets from the network (the `netperf`<sup>2</sup> benchmark has been used for this purpose, for

<sup>2</sup><http://www.netperf.org>

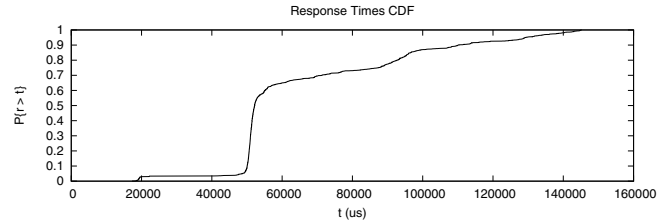


Figure 1. CDF of the response times for a  $\tau = (20ms, 50ms)$  real-time task in a standard Linux kernel with high network traffic.

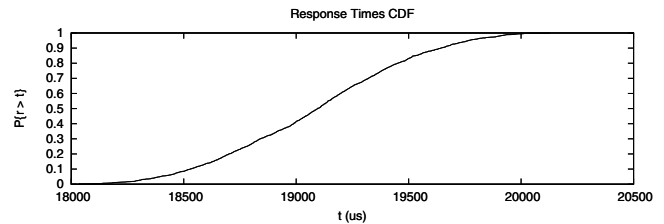


Figure 2. CDF of the response times for a  $\tau = (20ms, 50ms)$  real-time task in a Preempt-RT kernel with high network traffic and  $\tau$ 's priority higher than the IRQ thread priority.

it generates the desired workload and evaluates the network throughput).

When the real-time task runs alone on an unmodified Linux kernel, the maximum measured response time is around  $20ms$  (corresponding to the WCET) as expected. When running the network server alone on an unmodified Linux kernel and using small packets (around 192 bytes), the measured network throughput is around  $74Mbps$ . However, when the two tasks are executed simultaneously the worst-case response time of the real-time task grows to more than  $150ms$  (even if the program receiving UDP packets is executed as a non real-time task), as shown in Figure 1 (representing the Cumulative Distribution Function - CDF - of the response times).

When executing the two tasks on a preempt-RT kernel, it is possible to set the priority of the IRQ threads so that the real-time guarantee for  $\tau$  is not jeopardised: if the priority of the IRQ threads is less than  $\tau$ 's priority, then the worst-case response time is still very close to  $20ms$  (see Figure 2). However, the network throughput drops to  $48Mbps$ . As one would expect, the network throughput returns to  $74Mbps$  if the priority of the network IRQ threads is raised to a value higher than the priority of  $\tau$ , but the response time of the latter is in this case out of control. These results confirm the outcome of some previous work [2] in showing that fixed priorities do not allow to find good trade-offs between the real-time performance of user-space tasks and the throughput of hardware devices.

#### B. Reservation-Based Scheduling of Interrupt Threads

The work presented in this paper addresses the problem of finding good latency/throughput trade-offs by using

reservation-based scheduling for the IRQ threads (in particular, a CBS with hard reservation behaviour is used).

The CBS parameters  $Q_i^s$  and  $T_i^s$  must be properly dimensioned to avoid dropping too many interrupts. For example, some hardware devices have a limited amount of memory for incoming data, and interrupt handlers are used to get data from this buffer. If too many interrupts are pending, the buffer can overflow and some input data can be lost (in the next sections, a network card which can buffer at most  $N_c$  input Ethernet frames will be considered). Hence, the CBS used to schedule the IRQ thread for the device must be dimensioned to control the number of pending interrupts.

A typical strategy for dimensioning a reservation is to choose a reservation period  $T^s$  smaller than the minimum inter-arrival time of the served task. Such a choice allows one to model a reservation as a queue and to use standard tools for Markov Chain analysis to compute the probability distribution of the response time. Unfortunately, this approach is not viable in our setting: indeed, two interrupts can even be separated by an interval of time of a few microseconds. On the other hand, choosing a value for  $T^s$  smaller than  $1ms$  leads to an unacceptable scheduling overhead. As discussed next, this consideration induces a different modelling technique than the one typically used in these cases.

#### IV. THEORETICAL ANALYSIS

In this section, the problem of correctly dimensioning the CBS parameters for serving an IRQ thread is addressed, starting with a deterministic model and then providing a stochastic analysis of the system. As already explained in Section III, thanks to the temporal isolation property provided by the CBS it is possible to analyse a single IRQ thread in isolation; hence, the analysis can be developed considering one single interrupt type.

##### A. Deterministic Analysis

In the deterministic case, which is simpler to analyse, interrupts are assumed to fire with a minimum inter-arrival time  $P$ , and the interrupt handler is assumed to need a maximum time  $C$  to serve an interrupt request. Based on such assumptions, the worst case situation is represented by periodic interrupt requests (with period  $P$ ), with the interrupt handler serving each one of them in a constant time  $C$ .

In this situation, the IRQ thread can be modelled as a periodic task  $\tau = (C, P)$  and properly dimensioning a CBS  $(Q^s, T^s)$  so that no interrupt is lost is quite simple. For the sake of simplicity and without loss of generality,  $T^s$  is set as a multiple of  $P$ . In a server period  $n = \frac{T^s}{P}$  interrupts arrive, requiring  $\frac{T^s}{P}C$  units of time to be served. Hence, the CBS must reserve at least  $Q^s = \frac{T^s}{P}C$  unit of time every server period  $T^s$ . Moreover, there is an upper bound to the server period, due to the maximum number of pending interrupts  $N_c$  that can be queued. In fact, in the worst case the CBS can

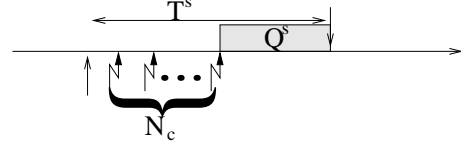


Figure 3. Worst case reservation behaviour for periodic interrupt arrivals.

provide the  $Q^s$  time units at the end of the server period, as shown in Figure 3; hence, in the first  $T^s - Q^s$  time units of the server period the IRQ thread will not execute, and  $\frac{T^s - Q^s}{P}$  pending interrupts will be queued. This number should be smaller than the maximum number of pending interrupts  $N_c$ .

As a result, the conditions expressed in Equation 1 must be satisfied:

$$\left\{ \begin{array}{l} \frac{Q^s}{T^s} \geq \frac{C}{P} \\ \frac{T^s - Q^s}{P} < N_c \end{array} \right. \quad (1)$$

As noticed above, this kind of conservative analysis can be applied even if the IRQ thread is not periodic (and/or its execution times are not constant), by substituting  $C$  with the WCET of the IRQ thread, and  $P$  with the minimum inter-arrival time between interrupts. However, assuming this kind of worst case conditions for the workload can be very pessimistic, often resulting in over-provisioning and in a suboptimal system utilisation.

To evaluate the degree of conservativeness of Equation 1 for real-world workloads, a stochastic model (offering a fine grained description of the system evolution) has been developed.

##### B. Stochastic Analysis

A stochastic analysis of a reservation-based system has already been performed in the past [6], [4], [20], but such previous approaches cannot be used in this context for the reasons explained at the end of Section III. Hence, a new stochastic model has been developed.

The model is constructed assuming that the inter-arrival time between the  $h^{th}$  and  $(h+1)^{th}$  interrupts is a stochastic process and that the time needed for serving the  $h^{th}$  interrupt is also a stochastic process. Both these stochastic processes are assumed to be independent and identically distributed (i.i.d.). However, the stochastic analysis presented in this paper differs from the previous ones because it is based on a discrete-time model, where all the times are multiple of a small time interval  $\Delta$ . The system evolution is necessarily observed at each time interval  $\Delta$  (this difference respect to the previous works allows to handle inter-arrival times smaller than  $T^s$  and to count the number of pending jobs).

Applying standard modelling techniques based on Discrete Time Markov Chains (DTMC)<sup>3</sup> it is possible to find the probability of dropping an interrupt. The state of the

<sup>3</sup>To be more correct, the resulting stochastic process is semi-Markovian.

IRQ thread can be described by a 5-tuple  $S_i = (x, h, z, t, q)$  where:

- $x$  is the number of the pending interrupts;
- $h$  is the time to the arrival of the next interrupt;
- $z$  is the residual execution time of the currently served interrupt;
- $t$  is the time elapsed from the beginning of the last server period;
- $q$  is the residual budget of the CBS.

Each variable is described by an integer number of time intervals  $\Delta$ , therefore the number of states is numerable. Since the system cannot queue more than  $N_c$  pending interrupts,  $x \leq N_c$ . All other variables are also bounded, hence the number of states is finite (i.e., model is finite).

The transitions between the states are driven by 1) the evolution of time (which has an impact on  $h, z, t$ ), 2) the arrival of a new interrupt (that has an impact on  $x$ ), 3) the allocation of the CPU by the scheduler (which decreases  $q, z$  and, when the current task finishes, may determine a decrease of  $x$ ). The probability to execute a task served by a CBS at time  $t$  is estimated by Equation 2

$$Exec(q, t) := \begin{cases} 1 & \text{with probability } \frac{q}{T^s - t} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

(remember that  $q$  is the current budget and  $t$  is the time elapsed from the beginning of the period). At the beginning of a reservation period (when the deadline is postponed), the task has a probability to execute equal to  $Q^s/T^s$ . After 1 time unit, the execution probability becomes  $(Q^s - 1)/(T^s - 1)$  if the task executed, or  $Q^s/(T^s - 1)$  if the task did not execute. Note that if the task does not execute for  $T^s - q$  time units, then it is guaranteed to execute in the next time unit ( $Exec(q, t)$  gives an execution probability equal to 1). This function models the properties of a hard CBS, so that a task executes for  $Q^s$  time units every  $T^s$  time units.

The resulting set of states transitions is clearly very large and complex, and due to space constraints it is not possible to fully describe it. However, the most important transitions are described in the following.

The initial state of the system is  $S_0 = (0, 0, 0, 0, 0)$ . A set of states with  $x = -1$  is used to model the situation in which an interrupt is lost (an interrupt arrived when there already are  $N_c$  pending interrupts).

An interrupt is completely served (and the corresponding job of the IRQ thread finishes) when  $z$  arrives to 0; then, if there is a pending interrupt it can be served. Hence, a new job for the IRQ thread is generated, and its execution time is computed based on the Probability Distribution Function (PDF) of the service times  $U(c) = P\{c_{i,j} = c\}$ .

In the same way, a new interrupt fires (and becomes ready to be served or pending) when  $h$  arrives to 0; in this case, the arrival time of the next interrupt is computed based on the PDF of the inter-arrival times  $V(\delta) = P\{r_{i,j+1} - r_{i,j} = \delta\}$ .

The CBS budget is recharged at the next scheduling deadline (the end of the current server period) according to the hard reservation behaviour, so when  $t$  arrives to  $T^s$  its value is reset to 0 and  $q$  is recharged to  $Q^s$ .

As previously mentioned, the complete model is much more complex than this (there are many state transitions that have not been mentioned above), and the complete transition function between  $S_i$  and  $S_{i+1}$  is described in Figure 9, at the end of this paper.

The presented model permits to compute the probability to drop an interrupt, which is equal to the steady state probability of the  $(-1, h, z, t, q)$  states. Such probability can be computed by finding the steady state probabilities through some numerical tool: in practice, if  $\pi(n)$  is a vector containing the state probabilities at time  $n\Delta$  (that is, the  $i^{th}$  component of  $\pi(n)$  is probability that at time  $n\Delta$  the system is in the  $i^{th}$  state - remember that the states are numerable), then  $\pi(n+1) = \pi(n)M$ , where  $M$  is a transition matrix containing the values from the model presented in Figure 9. The searched steady state probabilities are given by  $\pi = \lim_{n \rightarrow \infty} \pi(n)$ , and can be found by numerically solving the equation  $\pi = \pi M$  (that is, finding the eigenvector with eigenvalue 1 for the matrix  $M$ ).

Although  $M$  can be a very large matrix, it has a sparse structure (the non-null terms are a small percentage of the elements of the matrix). Therefore, the computation is tractable appropriate tools for sparse matrix (in particular we used the sparse matrices libraries from the Trilinos Project<sup>4</sup>).

### C. Selected Results

The proposed model has been numerically solved using different PDFs for the execution and inter-arrival times, and different  $(Q^s, T^s)$  settings, verifying that the results are consistent with Equation 1.

A first set of results shows that if the PDFs of the execution and inter-arrival times are deterministic, then the results exactly match the deterministic analysis. To achieve this result, the PDF of the execution times has been set to  $U(2) = 1$ , and the PDF of the inter-arrival times has been set to  $V(4) = 1$  (hence, interrupts are periodic with period  $P = 4$  and each interrupt needs 2 time units to be served). Then, when solving the eigenvector problem mentioned above it turns out that the probability to drop an interrupt is  $> 0$  if  $Q^s/T^s < 0.5$ . If, instead,  $Q^s/T^s \geq 0.5$ , things are more interesting and the probability to drop an interrupt depends on the period. Consistently with Equation 1, if  $Q^s > T^s - N_c P = T^s - N_c \cdot 4$  then such probability is 0. For example, Figure 4 shows the probability to drop an interrupt when  $T^s = 2Q^s$  and  $N_c = 4$  and it is possible to see that interrupts are not lost for  $Q^s < 2Q^s - 4 \cdot 4 \Rightarrow Q^s < 16$ .

The effect of the maximum number of pending interrupts  $N_c$  is also shown in Figure 5, which plots the interrupt loss

<sup>4</sup><http://trilinos.sandia.gov>

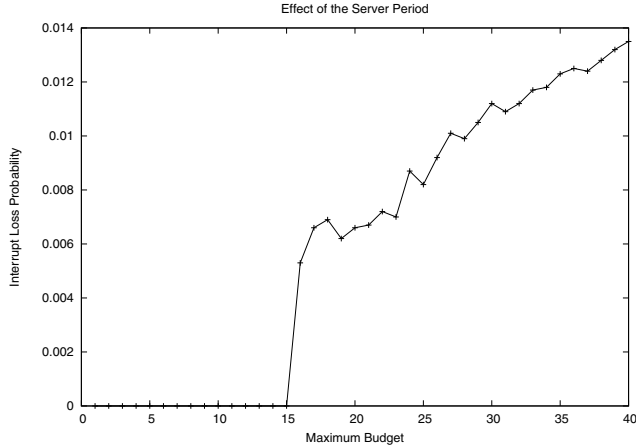


Figure 4. Interrupt loss probability for  $P = 4$ ,  $C = 2$ ,  $N_c = 4$  and  $T^s = 2Q^s$  as a function of  $Q^s$ .

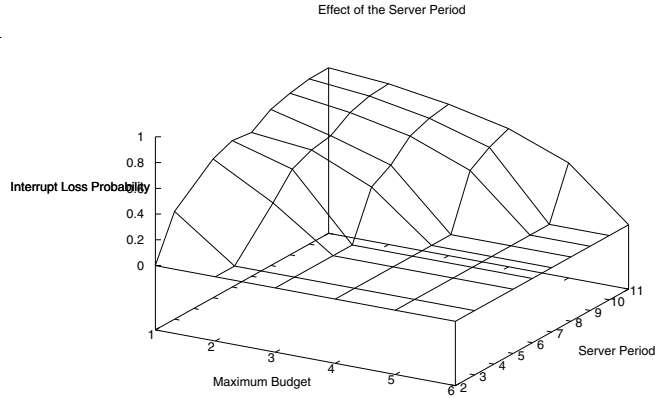


Figure 6. Interrupt loss probability for  $P = 4$ ,  $C = 2$ , and  $N_c = 32$  as a function of  $(Q^s, T^s)$ .

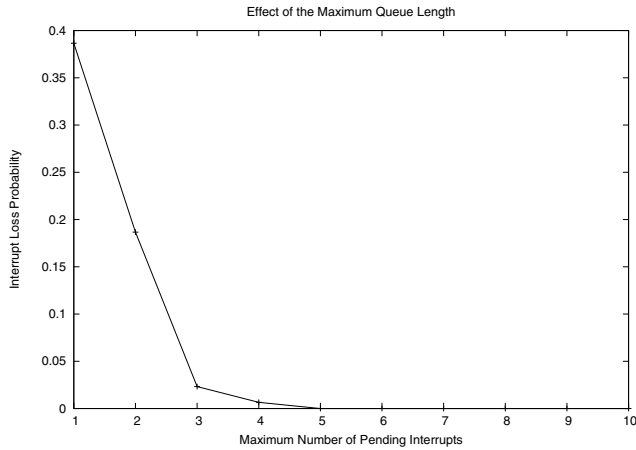


Figure 5. Interrupt loss probability for  $P = 4$ ,  $C = 2$ ,  $Q^s = 20$ , and  $T^s = 40$  as a function of  $N_c$ .

probability for a  $(20, 40)$  CBS as a function of  $N_c$ . Finally, Figure 6 shows how the interrupt loss probability depends on both  $Q^s$  and  $T^s$ .

The advantages of using a stochastic model become even more visible when execution and inter-arrival times are not fixed. For example, consider a simple example with  $V(\delta) : V(4) = 0.2, V(5) = 0.8$  and  $U(c) : U(2) = 0.3, U(3) = 0.7$  (hence, the maximum load is  $3/4 = 0.75$ ). The maximum amount of pending interrupts is assumed to be  $N_c = 3$ . According to the deterministic analysis, the reserved fraction of CPU time must be  $Q^s/T^s > 0.56250$ , and the maximum budget must be  $Q^s > T^s - 3 \cdot 4 = T^s - 12$ . However, Table I shows that even if these two conditions are not respected the probability to drop an interrupt can be very low.

## V. IMPLEMENTATION AND EXPERIMENTS

An implementation of the CBS scheduler for Linux [17] has been used to schedule the IRQ threads in the 2.6.21.4-

Table I  
PROBABILITY TO DROP AN INTERRUPT FOR VARIOUS  $(Q^s, T^s)$  ASSIGNMENTS.

$Q^s$	$T^s$	$\frac{Q^s}{T^s} \geq 0.75$	$Q^s \geq T^s - 12$	$P\{\text{Drop}\}$
3	4	Yes	Yes	0.0
6	8	Yes	Yes	0.0
9	12	Yes	Yes	0.0
12	16	Yes	Yes	0.0
15	20	Yes	Yes	0.0
18	24	Yes	Yes	0.0
21	28	Yes	Yes	0.0
2	4	No	Yes	0.0027
4	8	No	Yes	0.0062
6	12	No	Yes	0.0102
8	16	No	Yes	0.0142
10	20	No	Yes	0.0215
12	24	No	No	0.0244
14	28	No	No	0.0269
4	6	No	Yes	$4.11e - 16$
8	12	No	Yes	$2.12e - 15$
12	18	No	Yes	0.0058
16	24	No	Yes	0.0056
20	30	No	Yes	0.0055
24	36	No	No	0.0054
4	7	No	Yes	$2.011e - 08$
8	14	No	Yes	$5.29e - 07$
12	21	No	Yes	0.0064
16	28	No	No	0.0060
20	35	No	No	0.0056
24	42	No	No	0.0063

rt12-cfs-v17 of the RT-Preempt for the Linux kernel. This patch transforms both the ISRs and the bottom halves in kernel threads (the hard IRQ threads and the soft IRQ threads), and the CBS scheduler is used to handle such threads (in the experiments reported in this section, a hard reservation behaviour has been used, to leave some execution time to non-real-time userspace tasks).

The experiments reported in this section focus on the network card as a hardware device, because it is a good example of a high-throughput device (up to 100Mbps) for which it is possible to control the input load (this is needed

Table II  
PDF OF THE INTER-PACKET TIMES.

$\delta(\mu s)$	$P\{r_{i+1} - r_i = \delta\}$
100	0.312393
110	0.684386
120	0.001791
130	0.000210
140	0.000240
150	0.000290
160	0.000420
170	0.000090
180	0.000070
190	0.000110

Table III  
PROBABILITY TO DROP AN INTERRUPT, ACCORDING TO THE THEORETICAL MODEL AND TO EXPERIMENTAL MEASUREMENTS.

$Q^s(\mu s)$	$T^s(\mu s)$	Experimental Results	Stochastic Model
100	1000	0.55	0.5899
200	1000	0.18	0.1997
300	1000	0	0
1000	10000	0.63	0.6717
2000	10000	0.27	0.3105
3000	10000	0	0.0007

for checking the consistence between the theoretical analysis and the experimental results). In particular, the test system used a “National Semiconductor Corporation DP83815” ethernet card (handled by the “natsemi dp8381x” Linux driver), which has an input ring buffer able to store up to 32 ethernet frames. This means that at most 32 ethernet IRQs can be simultaneously pending without dropping any input packet ( $N_c = 32$ ). The test system is based on an Intel Pentium 4 CPU running at 1700MHz, equipped with 256MB of RAM.

To generate a controlled load on the network card, the test machine has been connected to a traffic generator through a cross network cable. The traffic generator is another PC based on RT-Preempt, which is able to send UDP packets according to a specified distribution of the inter-packet times.

#### A. Validation of the Theoretical Model

First of all, the theoretical analysis developed in Section IV has been validated through a set of experiments, by using the traffic generator to send fixed-size UDP packets to the test system according to a specified PDF of the inter-packet times, shown in Table II. The amount of time needed by the IRQ thread to receive an ethernet frame has then been measured by using accurate CPU accounting and measuring the execution time for receiving a large number of ethernet frames. A large number of runs revealed that the amount of CPU time needed by the IRQ thread to receive an ethernet frame does not depend on the frame size and is between  $22\mu s$  and  $25\mu s$ .

Then, the number of network card IRQs in the test machine has been measured by reading `/proc/interrupts`

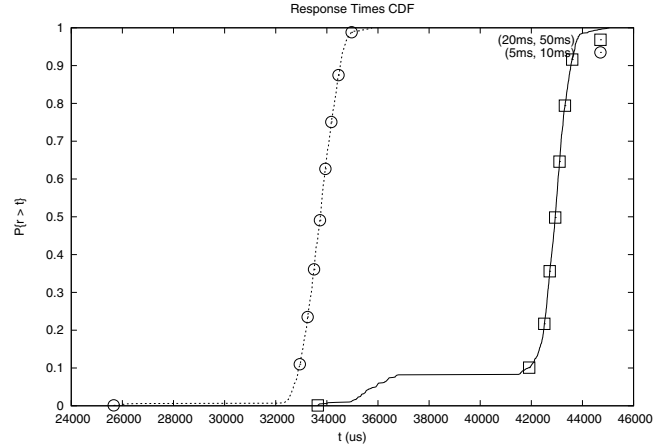


Figure 7. CDF of the response times for a  $\tau = (20ms, 50ms)$  real-time task in a standard Linux kernel with high network traffic, when serving  $\tau$  and the IRQ thread with CBS.

(to check if it matched the number of ethernet frames sent by the traffic generator), and the number of correctly received packets has been measured by using the `netstat` command. Based on these two measurements, it has been possible to compute the fraction of lost ethernet frames (equal to the fraction of dropped ethernet IRQs), displayed in the second column of Table III.

The stochastic model has then been used to compute the packet loss probability (using a PDF of the execution times consistent with the numbers measured above), and the results are displayed in the third column of Table III. These results show that there is a good match between the theoretical model and the experiments, and that the stochastic model is pessimistic (it always gives a probability to drop an interrupt that is slightly higher than the one measured in the real experiments). This last property is important for providing stochastic real-time guarantees. Note that the deterministic model can be used by considering the minimum inter-arrival time between interrupts ( $100\mu s$ ) and the maximum interrupt service time ( $25\mu s$ ). The results obtained by using Equation 1 indicate that only the  $(300\mu s, 1000\mu s)$  configuration results in no lost interrupt, but do not provide any information about the amount of lost interrupts. On the other hand, the stochastic model shows that some other configurations can also be useful (for example,  $(3000\mu s, 10000\mu s)$ ).

#### B. Back to the Motivational Example

The analysis presented in the previous section can be used to assign the reservation parameters so that the problem with `netperf` and small network packets showed in Section III-A is solved. After estimating the PDF of the inter-arrival times of the packets generated by `netperf` and of the computation time, we used the stochastic model to compute the packet loss probability when a  $(4ms, 10ms)$

CBS is used for the network hard IRQ thread. The resulting probability was 0. The experiments confirmed this result, showing a network throughput of  $74\text{Mbps}$  (which is equal to the network throughput measured when the network hard IRQ thread is scheduled with the maximum fixed priority). In the same experiment, we used a  $(20\text{ms}, 50\text{ms})$  CBS to serve the periodic real-time task, and the worst case response time has been measured as  $46\text{ms}$  (which is consistent with the expectations: since  $Q^s = 20\text{ms}$  is larger than the WCET, the worst case response time is expected to be less or equal than  $T^s = 50\text{ms}$ [4]).

The worst case response time of the periodic task can be reduced by using a smaller server period and reserving a larger fraction of the CPU time to the periodic task. To verify this hypothesis, the experiment has been repeated scheduling the periodic task with a  $(5\text{ms}, 10\text{ms})$  CBS, and using a  $(2\text{ms}, 10\text{ms})$  CBS for the IRQ thread (a fraction of the CPU time has been left unused for the `netperf` server, or for non real-time activities or other IRQ threads). The stochastic model allowed to compute a probability to drop an interrupt equal to 17.8%, and the throughput measured in the experiment was  $65\text{Mbps}$  (a drop rate of about 12% w.r.t. the previous  $74\text{Mbps}$ ). The difference between the expected 17.8% and the measured 12% is probably due to the slight divergence of the IRQ thread from the theoretical model, since it used to serve more than one interrupt at time, with a polling mechanism. The expected worst case response time was less than  $\lceil C/Q^s \rceil T^s = 40\text{ms}$ , and indeed a worst case response time of  $36\text{ms}$  has been measured.

Figure 7 shows the CDFs of the real-time task's response times measured in the two experiments described above (using a  $(20\text{ms}, 50\text{ms})$  CBS or a  $(5\text{ms}, 10\text{ms})$  CBS to serve the real-time task).

Summing up, the previous two examples show how the CBS can be used to find latency/throughput trade-offs, and the proposed theoretical model can be used to analyse the CBS parameters assignments. Practical experiments on real hardware show a reasonable consistence with the theoretical analysis.

### C. Controlling the System Performance

After verifying the correctness of the theoretical model, some additional experiments have been performed to verify that the proposed approach allows to control the real-time and I/O performance. This evaluation has been performed by first verifying that if  $\sum_i Q_i^s/T_i^s \leq 1$  then the worst case response times of a task  $\tau_i$  can be controlled by modifying its CBS parameters  $Q_i^s$  and  $T_i^s$ , and are not affected by the other tasks running in the system and by the I/O load. For this purpose, some periodic tasks  $\tau_i = (C_i, P_i)$  have been run, measuring the worst case response times. Then, `netperf` has been used to evaluate the network throughput between the traffic generator and the test system when the IRQ threads are scheduled through the CBS (and to check

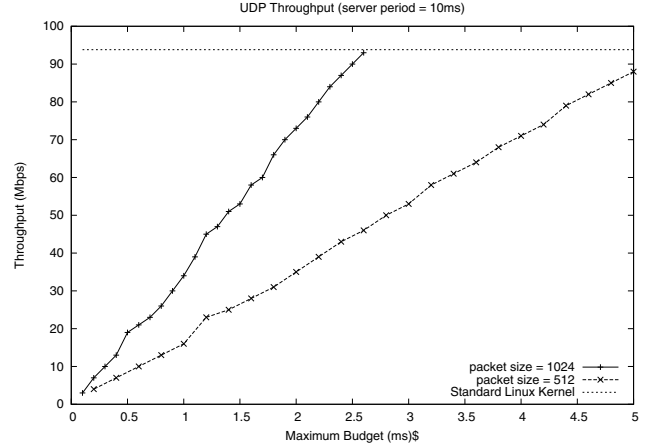


Figure 8. Network throughput as a function of the maximum budget  $Q^s$  ( $T^s = 10\text{ms}$ ).

Table IV  
MAXIMUM MEASURED INTER-PACKET TIMES AS A FUNCTION OF THE SCHEDULER.

CBS Parameters	Maximum Inter-Packet
$(1\text{ms}, 10\text{ms})$	$52593\mu\text{s}$
$(1.5\text{ms}, 10\text{ms})$	$42922\mu\text{s}$
$(2\text{ms}, 10\text{ms})$	$29869\mu\text{s}$
$(3\text{ms}, 10\text{ms})$	$489\mu\text{s}$
$(10\text{ms}, 100\text{ms})$	$323716\mu\text{s}$
$(20\text{ms}, 100\text{ms})$	$147064\mu\text{s}$
$(30\text{ms}, 100\text{ms})$	$491\mu\text{s}$
No CBS	$656\mu\text{s}$

that the worst case response times of the periodic tasks are not affected). The test focused on the network hard IRQ thread (running the ISR for the network card) and UDP traffic, and the results for 2 different packet sizes (1024 bytes and 512 bytes) are displayed in Figure 8.

From the figure it is possible to notice that when the reserved CPU time is enough the network throughput using IRQ threads and CBS is comparable with the network throughput of a standard Linux kernel (quite near to the theoretical maximum). Hence, the proposed approach does not affect the network performance too badly. It is also worth noting that the network throughput is linearly proportional to  $Q^s/T^s$ , and that the scaling factor depends on the packet size. This last result is consistent with the fact that each ethernet frame needs an almost-constant time to be processed, independently from the size, as previously measured (in particular, note that the slope of the curve for packet size = 512 is about half of the other curve, showing that the CPU time needed by the IRQ thread is about double).

In the following experiment, a stream of UDP packets has been sent from the traffic generator to the test system, with a fixed inter-packet time  $P = 100\mu\text{s}$ , measuring the times when the packets were received by a user-space real-time application running on the test system. The difference

between the reception times of two consecutive packets (expected to be  $100\mu s$ ) depends on the  $(Q^s, T^s)$  scheduling parameters of the IRQ thread. The results showed that if the fraction of CPU time reserved to the ethernet IRQ thread is not enough, then the times between the reception of two consecutive packets can be very large. But if enough CPU time is reserved, then the inter-packet times are very stable (more stable than for an unpatched Linux kernel). The maximum inter-packet times are reported in Table IV.

## VI. CONCLUSIONS AND FUTURE WORK

This paper presented the application of reservation-based scheduling (more specifically, the CBS algorithm) to interrupt threads. Two analysis techniques have been presented for correctly dimensioning the reservation parameters, and the analysis has been validated through a set of experiments on a real-time version of the Linux kernel.

As a future work, the stochastic model will be simplified (the current model is quite complex, and the number of states and transitions can probably be reduced), and a continuous-time model will be tested as an alternative. Moreover, it will be extended to analyse more complex situations (chains of interrupt threads, etc...). Additional experiments will also be performed using different hardware and multiple cooperating resources.

## REFERENCES

- [1] L. Kleinrock, *Queuing Systems*. Wiley-Interscience, 1975.
- [2] G. Modena, L. Abeni, and L. Palopoli, "Providing qos by scheduling interrupt threads," in *Work in Progress of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2008)*, St. Louis, April 2008.
- [3] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating systems support for multimedia applications," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, May 1994.
- [4] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.
- [5] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems," in *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.
- [6] L. Abeni and G. Buttazzo, "Stochastic analysis of a reservation-based system," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium.*, San Francisco, California, April 2001.
- [7] P. Druschel and G. Banga, "Lazy receiver processing (LRP): A network subsystem architecture for server systems," in *Proceedings of the second USENIX symposium on Operating Systems Design and Implementation (OSDI)*. Seattle, Washington, United States: [New York, NY, ACM Special Interest Group on Operating Systems], 1996, pp. 261–276.
- [8] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Transactions on Computer Systems*, vol. 15, August 1997.
- [9] J. Brustoloni, E. Gabber, A. Silberschatz, and A. Singh, "Signaled receiver processing," in *Proceedings of the USENIX Annual Technical Conference*, June 2000, pp. 211–224.
- [10] R. Black, P. Barham, A. Donnelly, and N. Stratford, "Protocol implementation in a vertically structured operating system," in *Proceedings of the 22nd Annual Conference on Local Computer Networks*, 1997.
- [11] Y.Zhang and R. West, "Process-aware interrupt scheduling and accounting," in *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, Rio de Janeiro, Brazil, December 2006.
- [12] H. Haertig, R. Baumgartl, M. Borriss, C. joachim Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schnberg, and J. Wolter, "DROPS - OS support for distributed multimedia applications," in *In Proceedings of the Eighth ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.
- [13] H. Haertig and M. Roitzsch, "Ten years of research on L4-based real-time systems," in *Proceedings of the Eighth Real-Time Linux Workshop*, Lanzhou, China, 2006.
- [14] K. Jeffay and G. Lamastra, "A comparative study of the realization of rate-based computing services in general purpose operating systems," in *Proceedings of the Seventh IEEE International Conference on Real-Time Computing Systems and Applications (RTCSA)*, Cheju Island, South Korea, 2000, pp. 81–90.
- [15] S. Ghosh and R. Rajkumar, "Resource management of the OS network subsystem," in *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2002.(ISORC 2002)*, 2002, pp. 271–279.
- [16] S. Rostedt, "Internals of the rt patch," in *Proceedings of the Linux Symposium*, Ottawa, Canada, June 2007.
- [17] L. Abeni and G. Lipari, "Implementing resource reservations in linux," in *Proceedings of Fourth Real-Time Linux Workshop*, Boston, MA, December 2002.
- [18] M. Lewandowski, M. Stanovich, T. Baker, K. Gopalan, and Wang, "Modeling device driver effects in real-time schedulability analysis: Study of a network driver," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, Bellevue, WA, 2007.
- [19] T. Baker, A. Wang, and M. J. Stanovich, "Fitting linux device drivers into an analyzable scheduling framework," in *Proceedings of the Workshop on Operating Systems Platforms for Embedded Real-time Applications*, Pisa, Italy, July 2007.
- [20] L. Abeni and G. Buttazzo, "Qos guarantee using probabilistic dealines," in *Proceedings of the IEEE Euromicro Conference on Real-Time Systems*, York, England, June 1999.

$(1, j, 0, 0, Q^S)$	$(0, 0, 0, 0, 0)$ with prob $V(j)$ (1)
$(0, h-1, 0, t+1, q)$	$(0, > 1, z, t, q)$ (2)
$(N_c-1, h-1, 0, t+1, q-1)$	$(-1, > 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (4)
$(N_c, h-1, z, t+1, q)$	$(-1, > 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (5)
$(-1, j, z-1, t+1, q-1)$	$(-1, 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (6)
$(-1, j, z, t+1, q)$	$(-1, 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (7)
$(N_c, h-1, z-1, t+1, q-1)$	$(-1, > 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (8)
$(N_c, h-1, z, t+1, q)$	$(-1, > 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (9)
$(N_c, j, 0, t+1, q-1)$	$(-1, 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (10)
$(-1, j, z, t+1, q)$	$(-1, 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (11)
$(N_c-1, h-1, 0, t+1, q-1)$	$(-1, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)$ (12)
$(N_c, h-1, k-1, t+1, q-1)$	$(-1, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)$ (13)
$(N_c, h-1, z, t+1, q)$	$(-1, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (14)
$(N_c, j, 0, t+1, q-1)$	$(-1, 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)V(j)$ (15)
$(-1, j, k-1, t+1, q-1)$	$(-1, 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)V(j)$ (16)
$(-1, j, z, t+1, q)$	$(-1, 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (17)
$(N_c-1, h-1, 0, 0, Q^S)$	$(-1, > 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (18)
$(N_c, h-1, z, 0, Q^S)$	$(-1, > 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (19)
$(-1, j, z-1, 0, Q^S)$	$(-1, 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (20)
$(-1, j, z, 0, Q^S)$	$(-1, 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (21)
$(N_c, h-1, z-1, 0, Q^S)$	$(-1, > 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (22)
$(N_c, h-1, z, 0, Q^S)$	$(-1, > 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (23)
$(N_c, j, 0, 0, Q^S)$	$(-1, 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (24)
$(-1, j, z, 0, Q^S)$	$(-1, 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (25)
$(N_c-1, h-1, 0, 0, Q^S)$	$(-1, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)$ (26)
$(N_c, h-1, k-1, 0, Q^S)$	$(-1, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)$ (27)
$(N_c, h-1, z, 0, Q^S)$	$(-1, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (28)
$(N_c, j, 0, 0, Q)$	$(-1, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)V(j)$ (29)
$(-1, j, k-1, 0, Q)$	$(-1, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)V(j)$ (30)
$(-1, j, z, 0, Q)$	$(-1, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (31)
$(x-1, h-1, 0, 0, Q^S)$	$(N_c, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)$ (32)
$(x, h-1, k-1, 0, Q^S)$	$(N_c, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)$ (33)
$(x, h-1, z, 0, Q^S)$	$(N_c, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (34)
$(x, j, 0, 0, Q^S)$	$(N_c, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)V(j)$ (35)
$(-1, j, k-1, 0, Q^S)$	$(N_c, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)V(j)$ (36)
$(-1, j, z, 0, Q^S)$	$(N_c, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (37)
$(x-1, h-1, 0, 0, Q^S)$	$(N_c, > 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (38)
$(x, h-1, z, 0, Q^S)$	$(N_c, > 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (39)
$(x, j, 0, 0, Q^S)$	$(N_c, 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (40)
$(-1, j, z, 0, Q)$	$(N_c, 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (41)
$(x, h-1, z-1, 0, Q^S)$	$(N_c, > 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (42)
$(x, h-1, z, 0, Q^S)$	$(N_c, > 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (43)
$(-1, j, z-1, 0, Q^S)$	$(N_c, 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (44)
$(-1, j, z, 0, Q^S)$	$(N_c, 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (45)
$(x, h-1, z, 0, Q^S)$	$(0, > 1, z, T^S-1, q)$ (46)
$(x+1, j, z, 0, Q^S)$	$(0, 1, z, T^S-1, q)$ with prob $V(j)$ (47)
$(x-1, h-1, 0, 0, Q^S)$	$(< N_c, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)$ (48)
$(x, h-1, k-1, 0, Q^S)$	$(< N_c, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)$ (49)
$(x, h-1, z, 0, Q^S)$	$(< N_c, > 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (50)
$(x, j, 0, 0, Q^S)$	$(< N_c, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)$ (51)
$(x+1, j, k-1, 0, Q^S)$	$(< N_c, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)V(j)$ (52)
$(x+1, j, z, 0, Q^S)$	$(< N_c, 1, 0, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (53)
$(x-1, h-1, 0, 0, Q^S)$	$(< N_c, > 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (54)
$(x, h-1, z, 0, Q^S)$	$(< N_c, > 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (55)
$(x, j, 0, 0, Q^S)$	$(< N_c, 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (56)
$(x+1, j, z, 0, Q^S)$	$(< N_c, 1, 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (57)
$(x, h-1, z-1, 0, Q^S)$	$(< N_c, > 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (58)
$(x, h-1, z, 0, Q^S)$	$(< N_c, > 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (59)
$(x+1, j, z-1, 0, Q^S)$	$(< N_c, 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (60)
$(x+1, j, z, 0, Q^S)$	$(< N_c, 1, > 1, T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (61)
$(x-1, h-1, 0, t+1, q-1)$	$(N_c, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)$ (62)
$(x, h-1, k-1, t+1, q-1)$	$(N_c, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)$ (63)
$(x, h-1, z, t+1, q)$	$(N_c, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (64)
$(x, j, 0, t+1, q-1)$	$(N_c, 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)V(j)$ (65)
$(-1, j, k-1, t+1, q-1)$	$(N_c, 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)V(j)$ (66)
$(-1, j, z, t+1, q)$	$(N_c, 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (67)
$(x-1, h-1, 0, t+1, q-1)$	$(N_c, > 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (68)
$(x, h-1, z, t+1, q)$	$(N_c, > 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (69)
$(x, j, 0, t+1, q-1)$	$(N_c, 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (70)
$(-1, j, z, t+1, q)$	$(N_c, 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (71)
$(x, h-1, z-1, t+1, q-1)$	$(N_c, > 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (72)
$(x, h-1, z, t+1, q)$	$(N_c, > 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (73)
$(-1, j, z-1, t+1, q-1)$	$(N_c, 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (74)
$(-1, j, z, t+1, q)$	$(N_c, 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (75)
$(x, h-1, z, t+1, q)$	$(0, > 1, 0, < T^S-1, q)$ (76)
$(x+1, j, z, t+1, q)$	$(0, 1, 0, > T^S - (\frac{q}{Q}T^S), q)$ with prob $V(j)$ (77a)
$(x+1, j, z, 0, Q)$	$(0, 1, 0, < T^S - (\frac{q}{Q}T^S), q)$ with prob $V(j)$ (77b)
$(x-1, h-1, 0, t+1, q-1)$	$(< N_c, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(1)$ (78)
$(x, h-1, k-1, t+1, q-1)$	$(< N_c, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}U(k)$ (79)
$(x, h-1, z, t+1, q)$	$(< N_c, > 1, 0, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (80)
$(x-1, h-1, 0, t+1, q-1)$	$(< N_c, > 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (81)
$(x, h-1, z, t+1, q)$	$(< N_c, > 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (82)
$(x, h-1, z-1, t+1, q-1)$	$(< N_c, > 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}$ (83)
$(x, h-1, z, t+1, q)$	$(< N_c, > 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}$ (84)
$(x+1, j, z-1, t+1, q-1)$	$(< N_c, 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (85)
$(x+1, j, z, t+1, q)$	$(< N_c, 1, > 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (86)
$(x, j, 0, t+1, q-1)$	$(< N_c, 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 1\}V(j)$ (87)
$(x+1, j, z, t+1, q)$	$(< N_c, 1, 1, < T^S-1, q)$ with prob $P\{Exec(q, t) = 0\}V(j)$ (88)

Figure 9. The stochastic model.