Can Real-Time Systems be Chaotic?

Lothar Thiele and Pratyush Kumar
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology Zurich, Switzerland

***Abstract*—In this paper, we take a dynamical systems perspective of real-time systems. In particular, we investigate the evolution of response times of periodic jobs and aim to show that oscillatory and chaotic behavior can be exhibited by standard scheduling algorithms. To this end, we present a simple periodic task specification that leads to oscillations of response times for a fixed priority scheduler. We then show three task specifications that lead to complex dynamic behavior under various scheduling algorithms: (a) round robin, (b) multiprocessor fixed-priority, and (c) priority inheritance protocol. As a practical validation of the results, we implemented the multiprocessor fixed-priority scheduler using POSIX threads and standard locking mechanisms. Finally, we discuss general observations and implications of the observed and proven phenomena.**

I. Introduction

The study of real-time systems mainly focuses on analyzing the lower and upper bounds on the response times for different scheduling scenarios such as round robin, fixed priority and priority inheritance in single- and multi-processor systems. This is practically motivated as such analysis helps to ascertain that deadlines are met and buffers do not overflow or underflow. Significant research questions are how to obtain hard bounds, and how to design systems to limit such bounds.

Response times of jobs often depend on what has been happening in the past. Some of the properties influencing the response times are for example queued jobs, the order of arrival of jobs, and the blocked resources. Under this view, we can think of a real-time system as some *dynamical system* where the quantities of interest are the response times of jobs of a task. A fixed *evolution rule*, dependent on the scheduling algorithm and the task models, describes how the response times vary across jobs. This variation is affected by the *state* of the system, which consists of properties such as the length of the pending queue, the associated remaining computing times of jobs, and other scheduler states. For a non-stochastic real-time system, given the initial conditions and future job arrivals, we can determine the evolution of all future response times.

In light of the above perspective, the standard response time analysis of real-time systems can be viewed as finding extrema on the set of all future response times. In addition to the extrema, we can also study other properties of the set of observed response times and their evolution over time. In particular, we can ask questions like: Do response times reach a fixed point, do they show periodic behavior, or is the evolution completely erratic, i.e., chaotic. Such questions have generated significant research interest in diverse disciplines such as geology, mathematics, microbiology, biology, economics, engineering, finance, meteorology, physics, and population dynamics. Driven by curiosity, we have been interested to see whether known phenomena of other dynamical systems can be observed in real-time systems in order to deepen our understanding of the class of systems we are dealing with, and to classify the properties they can exhibit. At this moment in time one can only speculate whether the surprising new insights eventually lead to relevant practical implications as it has been the case in some of the above mentioned areas.

One particularly interesting behavior of certain classes of dynamical systems is that of *chaos*. In this case, deterministic evolution rules of the system states lead to seemingly erratic behavior. These systems are highly sensitive to initial conditions which can be interpreted as (Edward Lorenz): "... the present determines the future, but the approximate present does not approximately determine the future." Chaotic behavior is often found in systems with non-linear dynamics and feedback mechanisms. One may remain hopeful about identifying chaotic behavior in real-time systems as the evolution of response times can be non-linear. A first example is the occurrence of discontinuities in response times of a preemptive fixed priority scheduler. As already discussed, feedback mechanisms are also found in real-time systems: Response times of current jobs can affect the response times of future jobs. *In the paper, we aim to identify different scheduler algorithms and task settings which lead to complex behavior of response times.*

Related work: Thinking of real-time systems as dynamical systems is not new, see for example [1]. In addition there has been some work, spread in time and area of interest, which study complex behavior in performance of queueing systems. The study of chaotic properties of manufacturing process was first discussed in [2]. In [3], the authors present the *bucket brigade* method as the first case of chaotic behavior in discrete manufacturing processes. Different manufacturing networks with feedback which lead to chaotic behaviour was studied in [4]. In [5], the authors studied a simple queueing system where the rate of serving two queues is given by a control law dependent on the difference between their fill rates. They showed that the buffer fill values evolve chaotically if the control law is suitably chosen to be a non-linear function. This problem was further studied in [6], [7], [8]. We did not find any specific work on the analysis of chaotic response-time evolutions in real-time systems.

Main contributions: In order to explicitly model feedback in real-time systems, we make us of standard task models that allow to specify task dependencies (acyclic DAG model). New jobs are released after a specific delay upon the finish of certain other jobs. These new jobs can then interfere with subsequent jobs and affect their response times, thereby modeling feedback as required for complex dynamic behavior. Based on this standard specification mechanism, we study different scheduling algorithms. We show that with a fixed priority scheduler, we observe a periodic behavior of response times with large periods. With a round-robin scheduler, we demonstrate response time behavior that shows signs of chaos. We show that this system shows a gradation of periodic and chaotic behavior for different parameter values. Furthermore, self-similarity is empirically observed, i.e., the characteristic of the evolution of response times exhibits a pattern that repeatedly displays at every scale of parameter variations. Then, we study a multiprocessor scheduling system, where we demonstrate chaotic behavior as well, which follows an existing chaotic map called the gingerbreadman map. We implement this scheduler using POSIX threads and standard locking mechanisms on a multicore processor. The measured response times confirm exhibition of chaotic behavior. In addition, we study a

priority inheritance protocol and again demonstrate chaotic behavior as specified by the gingerbreadman map. Finally, we describe some implications of complex dynamic behavior in real-time systems.

Organisation: We define the task model that will be used throughout the examples in Section II. In Section III we present the example of an oscillatory system with large periods. We give three examples of chaotic systems in Sections IV, V and VI. We finally conclude with some implications of the results in Section VII and conclusions in Section VIII.

II. Task Model

In this section, we define the task model and the associated notation, which we subsequently use in various scenarios. It is a standard model that allows to specify dependencies between tasks by means of acyclic graphs (DAG), see e.g., [9]. As feedback mechanisms are an important characteristic of complex dynamic systems, we specifically model situations where the completion of a job releases some other jobs after statically specified delays. We sometimes refer to these new jobs as "feedback" jobs, as they can potentially affect the timing behavior of the system after a certain delay. We illustrate such a task system with an example.

Example 1. *Consider a CAN bus that is serving all incoming and outgoing messages of an Electronic Control Unit (ECU). Whenever a measurement arrives from some external sensor, it is scheduled to be transmitted on the CAN bus. After the successful transmission to the ECU, the ECU computes a control value. This computed value is transmitted to an actuator through the same CAN bus.*

Thus, for the CAN bus as a resource, upon completion of a job (transmitting the sensor measurement), a new "feedback" job (transmitting the control value) is generated which is released after a specific delay (time for the ECU to compute).

The above example illustrates the creation of a single feedback job. More generally, multiple feedback jobs could be created after different delays upon the completion of the original job. In general, we will use the following periodic task model.

Definition 1. *A task system can be modeled as a set of acyclic directed task graphs $G = (V, E)$ where nodes represent tasks and edges represent dependencies. The release, finishing and response times of the i-th job of a task $v \in V$ are denoted as $A_v(i)$, $F_v(i)$ and $R_v(i) = F_v(i) - A_v(i)$, respectively. The execution time of the jobs of task $v \in V$ is denoted as C_v.*

Each task graph G has a single source node $s \in V$, i.e., a node without predecessors. To each task graph G there is associated a period P and a phase ϕ, which denote the period and the phase at which the jobs of its source task are released, i.e., $A_s(i) = iP + \phi$ for $i \geq 0$.

To each dependency $(v, w) \in E$ from node $v \in V$ to node $w \in V$ there is associated a delay $D(v, w)$: A new job of task w is released $D(v, w)$ time units after the finishing time of the corresponding job of task v, i.e., $A_w(i) = D(v, w) + F_v(i)$ for $i \geq 0$.

Depending on the specific scenario, the jobs of a task system can be scheduled according to different policies such as fixed priority, round-robin, fixed priority with priority inheritance or even executing on different processors. For instance in Example 1, messages to the actuator could have a higher priority on the fixed-priority CAN bus than messages from the sensor. In such settings, the execution of a job can affect the response time of other jobs from the same task, from other tasks, from the same period or from other periods. As we will show, these interferences lead to sometimes unexpected variability in the response times of jobs.

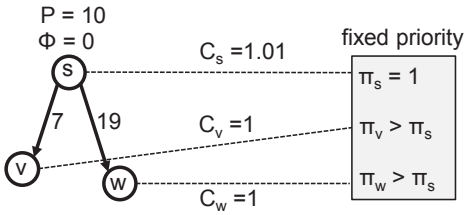

Fig. 1. Setting for fixed priority scheduling scenario.

The initial conditions of a task system will play a significant role in the forthcoming analysis. In some cases, we will need to specify properties of backlogged jobs that did not yet complete their execution but are part of the the initial system state, i.e., they exist on the system prior to the first job release. This information determines a significant part of the overall initial system state from which the dynamic evolution starts. We model this situation simply by listing the release times of earlier jobs of tasks. In other words, for some task v we specify a list $R_v = [R_v(-1), R_v(-2), \ldots]$, where $R_v(i)$ is the response time of the ith job with $i < 0$, i.e., before the first job $i = 0$ which we explicitly release.

In all scenarios described in the paper, we investigate the evolution of the response times of the periodically released source task $s \in V$ of one of the task graphs $G = (V, E)$ of a task system, i.e., we are interested in $R_s(i) = F_s(i) - (iP + \phi)$ for $i \geq 0$.

III. Fixed Priority Scheduling

In this section, we will present our first example of a real-time system with fixed-priority scheduling that shows complex behavior of its response times. In particular, the resulting system shows oscillatory behavior with periods which can be arbitrarily large.

The task setting is shown in Fig. 1. The tasks are scheduled using a fixed priority scheduler with priorities $\pi_s = 1$, $\pi_v > \pi_s$ and $\pi_w > \pi_s$ for tasks s, v and w, respectively, where low numbers denote low priority. There are no jobs in the system initially. The parameters are chosen such that previous jobs of tasks v and w can interfere with jobs of task s. Thus, the maximum interference is $C_v + C_w = 2$. Hence, the response times of the jobs of task s are in the interval $R_s(i) \in [1.01, 3.01]$. As mentioned before, we sometimes refer to the jobs of tasks v and w as feedback jobs as they are activated by jobs of task s and can affect the response times of jobs of task s.

A. Simulation Results

For the task parameters defined above, we simulate the schedule for the interval $0 \leq i \leq 500$. All simulations and numerical computations in the paper are performed symbolically using Mathematica [10] and thus do not have any rounding errors, which can be significant at high resolutions. The response times of the jobs of task s are plotted in Fig. 2. Notice how the response times change periodically within the specified bounds. It can be observed that there are 102 distinct values of the response times. Due to the limited resolution, this is not visible in Figs. 2 and 3.

We will now briefly discuss how such a variability is generated. The response time of the first job of task s equals its execution time, i.e., 1.01. The response time of the second job of s also equals 1.01 as the corresponding job of v does not interfere with it. The third job of s suffers interference from first job of w. The subsequent jobs of s start to suffer greater interference and their response time

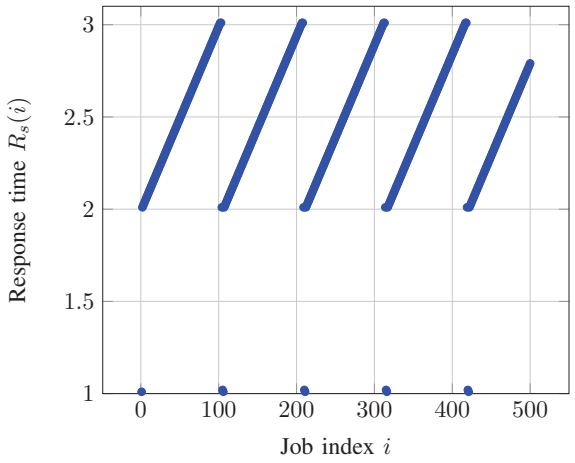

Fig. 2. Response times across jobs for the fixed priority scheduler.

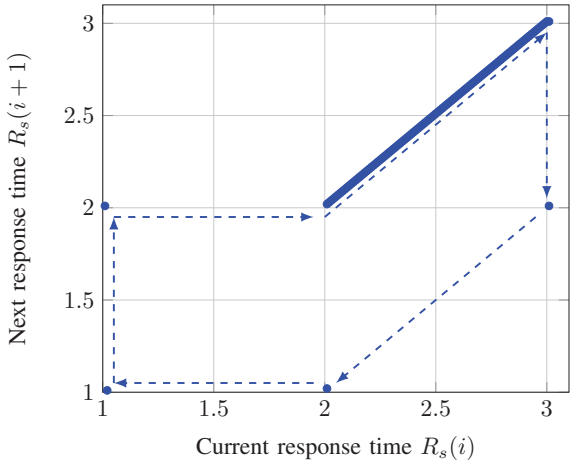

Fig. 3. Response time map for the fixed priority scheduler.

increases. Interestingly, as the response times of the jobs of s increase, the interference from jobs of v and w also increases linearly. This continues until the response time of a job of s increases to 3.01 when the response time of the next job of s decreases to 2.01, and subsequently to 1.01. This process repeats periodically.

B. Response Time Map

We now plot the set of all pairs $(R_s(i), R_s(i+1))$ for $i \geq 0$ in Fig. 3. First note that there is no fixed-point, i.e., there is no $R_s(i)$ such that $R_s(i+1) = R_s(i)$. Furthermore, the arrows illustrate the observed *periodic response time evolution*.

During the linear segment, the response time rises because of increasing interference from jobs of task v. This process halts when the response time rises enough to completely remove the interference from jobs of w. This then leads to the complete removal of the interference from jobs of v and the cycle repeats. Thus, the interplay between the arrival times of the feedback jobs, which in turn depend on finishing times of the jobs of s, leads to the interesting periodic behavior.

C. Remarks

In our first example, we have seen that sustained periodic changes to the response times can occur because of interference from dependent jobs. We look at some salient points of this example.

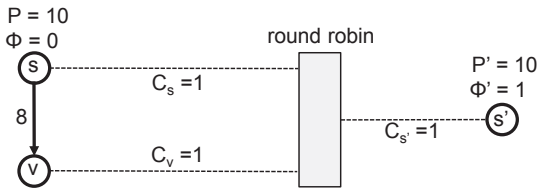

Fig. 4. Setting for the round robin scheduling scenario.

- The periodic evolution of response times is possible because of the absence of a fixed point in the response time map. We studied several examples of tasks with a single dependent job for the fixed priority scheduler. In all configurations, we found a fixed point and thus no periodic evolution. We believe that the constructed example with two dependent jobs is the minimal setting where large periodic evolutions of response times can be observed.
- The execution time $C_s = 1.01$ is responsible for the large period. A smaller value such as $C_s = 1.005$ would lead to even larger periods. This example illustrates that small deviations which will likely not have a large impact on the upper-bound of the response time can greatly affect the variation of the response time across jobs.

IV. Round Robin Scheduling

We consider now two tasks graphs, one consisting of two tasks s and v, the other one with a single task s'. We will study two different initial conditions as specified by the two initial response times $R_s(-1) = 1.1$ or $R_s(-1) = 1$. All tasks are scheduled according to round-robin as depicted in Fig. 4. We assume a small scheduling epoch, which fluidly divides the processing power. For instance, when two different jobs are queued, each job receives half the processing cycles per time unit. Similarly, when three different jobs are queued, each job receives one-third of it.

As an example, consider a shared bus with a bandwidth of 100 MB/s that arbitrates streams of packets according to round-robin. In a round-robin policy, time slices are assigned to each stream with pending packets in equal portions in a circular order. The bus services three sets of streams. Task s' represents stream s' that is a periodic stream with period 10 s, phase 1 s and a payload size of 100 MB. Stream s is identical but has a phase of 0 s. Once a packet of s is transmitted it is processed by a compute unit with a fixed execution time of 8s. The output of this compute unit is another stream v with a payload size of 100 MB. We are interested in the response time of packets from stream s.

The sample trace in Fig. 5 shows how the full bandwidth of the bus is distributed to the three streams as a function of time for $R_s(-1) = 1.1$. When a single stream has pending packets, it receives the full bandwidth of 100 MB/s. When two streams have pending packets, both receive 50 MB/s according to the round-robin policy under the assumption of negligibly small time-slices. Similarly, when three streams have pending packets, they all receive 100/3 MB/s each. Further, the arrival time of the packet with index i of stream v is 8 s later than the finish time of the packet with index i of s. As can be seen from the figure, different packets of s experience different response times due to variation in the amount of interference from the two other streams. Indeed, all the first five packets have different response times.

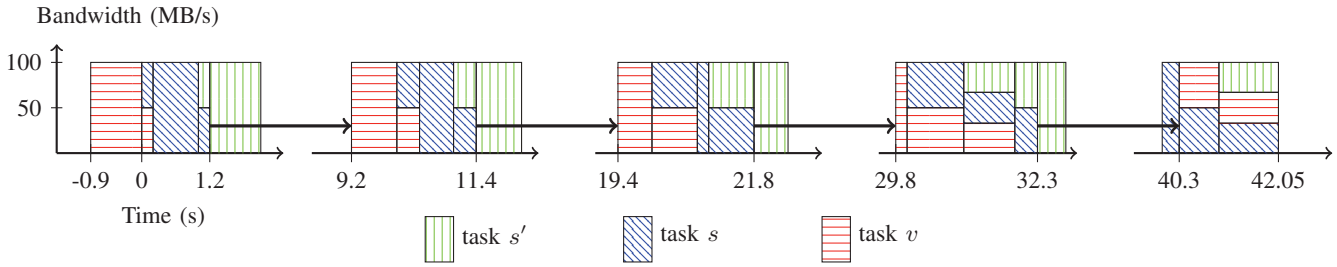

Fig. 5. Example trace from a round-robin scheduler. The observed response-times in time units of a second: $R_s(-1) = 1.1, R_s(0) = 1.2, R_s(1) = 1.4, R_s(2) = 1.8, R_s(3) = 2.3, R_s(4) = 2.05$.

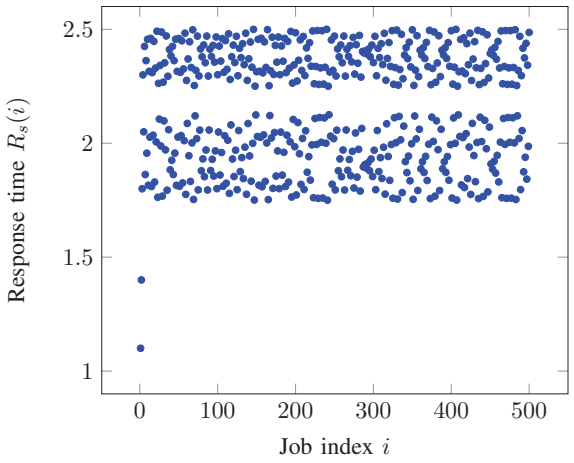

Fig. 6. Response times across jobs for the round robin scheduler with $R_s(-1) = 1.1$

.

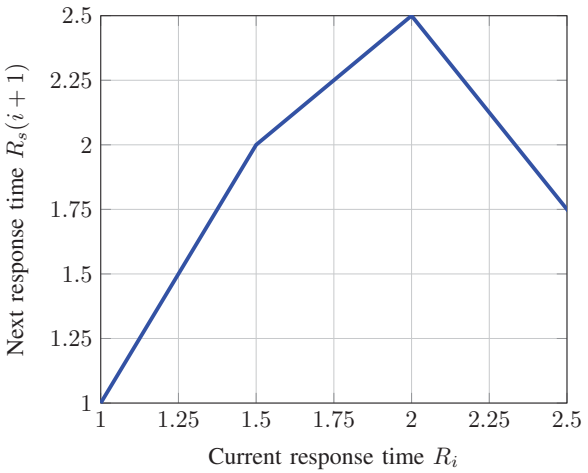

Fig. 7. Response time map for the round robin scheduler for $R_s(-1) = 1.1$.

A. Simulation Results

We study the response times of the jobs of s, where $R_s(i)$ denotes the response time of the ith job. We simulate the schedule for two cases of the initial conditions: (a) $R_s(-1) = 1$ and (b) $R_s(-1) = 1.1$. , respectively. For the first case, the response time of *all* jobs exactly equals 1. The response times of the first 500 jobs of s for $R_s(-1) = 1.1$ are shown in Fig. 6: The response times vary with no apparent periodicity. We will show later that this irregular variation shows properties of chaotic behavior. It can be shown that such a behavior will be observed whenever $R_s(-1)$ is greater than 1.

For the case when $R_s(-1) = 1$, at the release of the first job at time 0, there are no backlogged jobs. Hence, the job finishes by time 1. In turn, this implies that the job of v will not interfere the next job of s (job of task v finishes before the arrival of the next job). This process repeats, and thus we have a fixed point of the response time map of 1.

For the case when $R_s(-1) = 1.1$, the first job finishes at 1.2. This in turn causes a larger interference for the second job of s, which has a response time of 1.4. This evolution in response time is illustrated for the first five jobs in Fig. 5. This process continues leading to the chaotic behavior in the response times as shown in Fig. 6. It is remarkable, how the two very similar initial conditions lead to very different variations in the response times. For any initial condition with $R_s(-1) > 1$, we observe irregular variations in the response time.

B. Response Time Map

Since, the delay of the job of v is less than the period, we can compute the response time of a job given the finish times of the previous job. This is represented as the *response time map* M with $R_s(i+1) = M(R_s(i))$. From the analysis of the round robin scheduler, we obtain the response time map as shown in Fig. 7. Notice that 1 is a fixed point, i.e., $M(1) = 1$. This confirms the constant response time of 1 if $R_s(-1) = 1$. We will now prove a necessary condition for M being a chaotic map, see e.g., [11]:

Theorem 1. *The response time map M shown in Fig. 7 has sensitive dependence on initial conditions in its domain.*

Proof. A necessary condition for a map to be a chaotic is a *sensitive dependence on initial conditions* [11]. In other words, if we start from two very close but not the same points, and apply the map several times, then the two points must diverge, exponentially quickly. This property is equivalent to checking that the Lyaponuv exponent is positive. The Lyaponuv exponent, denoted λ, for a map M starting from some initial point x_0 is given as

$$\lambda = \lim_{p\to\infty} \frac{1}{p} \sum_{i=0}^{p-1} \ln |M'(x_i)|, \tag{1}$$

where

$$x_i = M(x_{i-1}), \forall\ i > 1 \tag{2}$$

and $M'(x)$ denotes the derivative of $M(x)$ w.r.t. x. We show that the map of Fig. 7 has a positive Lyaponuv exponent. The map comprises

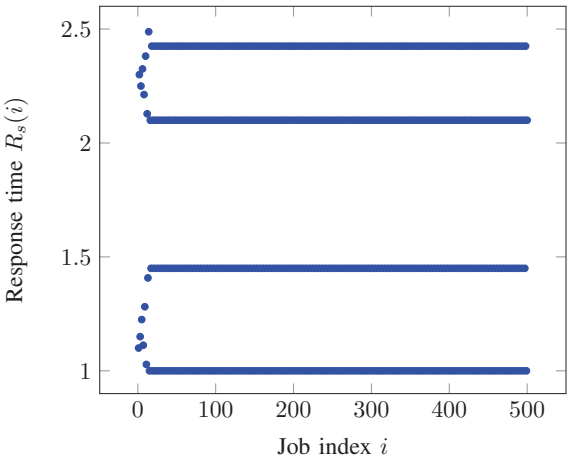

Fig. 8. Response time across jobs for the round robin scheduler with $D(s,v) = 8.6$ and $R_s(-1) = 1.1$.

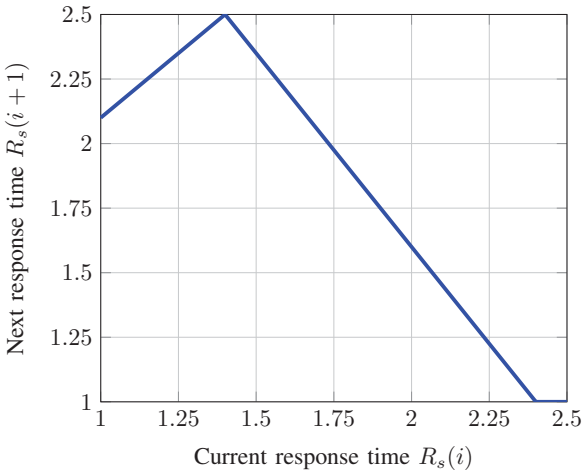

Fig. 9. Response time map for the round robin scheduler with $D(s,v) = 8.6$.

of 3 linear segments, none with a slope in the region $(-1, 1)$. Thus, in each term in the summation of (1) we have positive terms (logarithms of numbers with none lesser than 1 and some greater than 1).

In addition to the positive Lyaponuv exponent, the map M is closed under itself in the domain $\mathcal{D} = [1, 2.5]$, i.e., $M[\mathcal{D}] = \mathcal{D}$. This proves that M has sensitive dependence on initial conditions in its domain. □

The above theorem confirms that the variation observed in the response times in Fig. 6 shows a sensitive dependence on initial conditions.

C. The Delay Parameterisation

So far we have fixed the delay between the finishing of a job of s and the release of the corresponding feedback job of v to $D(s,v) = 8$. We now study the same setup for a different value of $D(s,v)$. In particular, consider $D(s,v) = 8.6$ with $R_s(-1) = 1.1$. For this case, the simulated response times are shown in Fig. 8. Clearly, the response times of the jobs change periodically (except for the initial few jobs) with only four unique response times. The corresponding response time map is shown in Fig. 9. Thus, while we observed properties of chaotic behavior with $R_s(-1) = 1.1$ for $D(s,v) = 8$, a change to $D(s,v) = 8.6$ makes the behavior periodic with a small period of 4.

The variation of the response times is thus a function of the delay parameter. To study this more closely, we compute the response time map with a parameter δ where $D(s,v) = 8 - \delta$. Again by performing response time analysis for the round robin scheduler we obtain the following map with δ as a parameter:

$$R_s(i+1) = M_\delta(R_s(i)) \tag{3}$$

where,

$$\begin{aligned} M_\delta(x) &= 2 \cdot (x - \delta) - 1, && 1 \le x \le 3/2 + \delta, \\ &= 1/2 + (x - \delta), && 3/2 + \delta < x \le 2 + \delta, \\ &= 11/2 - 3/2 \cdot (x - \delta), && 2 + \delta < x < 3 + \delta, \\ &= 1, && 3 + \delta \le x \le 5/2. \end{aligned} \tag{4}$$

The above function may be understood as follows. It consists of at most four piece-wise linear segments. Three of these with slopes 2, 1, and -3/2 are represented in Fig. 7. The δ parameter (which is a negative number) shifts the plot of Fig. 7 to the left by $-\delta$. If $\delta < -0.5$, then at the end of the domain, we have the fourth linear segment with a slope 0 and the constant response time of 1. This can be verified with the map of Fig. 9. To obtain this map, we left-shift the map of Fig. 7 by 0.6 (as $\delta = -0.6$), and then insert a zero slope segment at the end of length 0.1 (=$-\delta - 0.5$).

For $\delta = -0.6$, we observed periodic variation in the response time. We now show that one of the conditions for chaotic behavior is not satisfied for $\delta < -0.5$.

Theorem 2. *For $\delta < -0.5$, the map of (4) has non-sensitive dependence on initial conditions in part of its domain.*

Proof. Let us define $\mathcal{S} = (3 + \delta, 5/2] \subseteq \mathcal{D}$. Then from (4) we conclude that $M_\delta(x) = 1$ for all $x \in \mathcal{S}$. Therefore, $M_\delta^n(x) = M_\delta^n(y)$ for all $n \ge 1$ and for all $x, y \in \mathcal{S}$. Therefore, there exist $x, y \in \mathcal{D}$ such that $\|x - y\| > 0$ but $\|M_\delta^n(x) - M_\delta^n(y)\| = 0$ for all $n \ge 1$. □

D. Bifurcation Diagram

We have seen that the map of (4) is sensitive on initial conditions in its domain for some values of δ and not for others. A standard way of representing such a dependence is the *bifurcation diagram*. In such a diagram we vary a parameter and plot the unique values in the evolution of response times. The number of such unique values is an upper-bound of the periodicity. For the map of (4), we plot the bifurcation diagram in Fig. 10.

For small values of δ, around -0.75, we have a very small period. As the value of δ increases, we see a larger number of unique response times. In particular we see the common phenomenon of period doubling [11], i.e., at certain specific values of the parameter, the period doubles as new unique values emerge. For δ close to -0.5, we have very large periods, ultimately leading to the observed behavior for $\delta \ge -0.5$.

We also were interested in observing self-similarity which is a common phenomenon observed in many bifurcation maps [11]. A self-similar object has exactly or approximately the same shape as one or more of its parts. To this end, we zoomed into the bifurcation map for specific ranges of the parameter δ. In particular, we consider two ranges $\mathcal{R}_1 = [-0.57, -0.54]$ and $\mathcal{R}_2 = [-0.5526, -0.55412]$. Note the relative lengths of these ranges. The length of $\mathcal{R}_1$ is about 11% of the plot in Fig. 10, while the length of $\mathcal{R}_2$ is about 5% of the length of $\mathcal{R}_1$. Also note that $\mathcal{R}_2$ is contained in $\mathcal{R}_1$. For both these

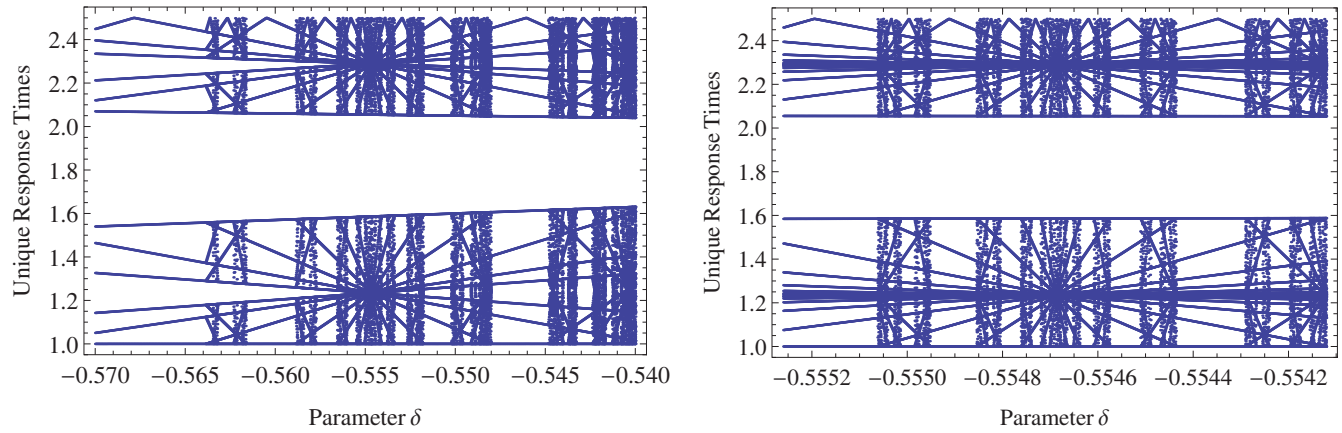

Fig. 11. Bifurcation diagrams for different ranges of the parameter δ for the map of (4).

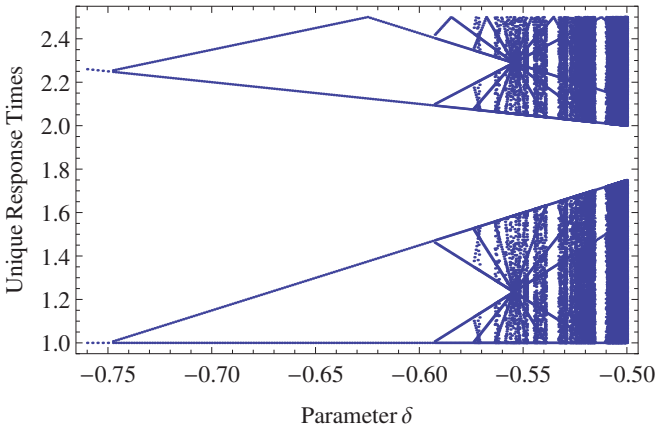

Fig. 10. Bifurcation diagram for the round robin scheduler.

ranges, of the parameter, we plot the bifurcation diagram in Fig. 11. The plots show a striking similarity. We observed similar patterns on zooming into other smaller regions of the parameter range.

E. Remarks

- To the best of our knowledge, the presented setting of round robin scheduling is the first example of a standard scheduling algorithm that shows signs of chaotic behavior. The response times of the jobs of task s vary irregularly and are sensitive to initial conditions, in spite of the seemingly simple specification.
- To the best of our knowledge, the family of maps given in (4) for $\delta \geq -0.5$, has not been observed in any other setting before.
- The bifurcation diagram with δ as a parameter is interesting from the study of dynamic systems. It shows many interesting properties of chaotic systems including period duplication, self-similarity and periodic windows (regions of the parameter space where the period suddenly reduces such as in [-0.548, -0.545] in Fig. 11).
- A thorough formal investigation of the new map given in (4) is beyond the scope of the paper.

V. Multi-Processor Scheduling

After considering single processor scenarios in previous sections, we now investigate tasks executing on multiple processors and

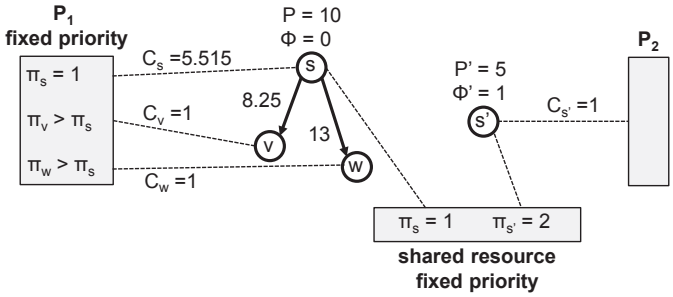

Fig. 12. Setting for the multi processor scenario.

sharing resources. In particular, we present examples of scheduling systems which exhibit chaotic variations in response times.

A. Setting

The task setting is shown in Fig. 12. In addition to the tasks s, v and w with the shown dependencies and their fixed priority scheduling on a first processor P_1, we have another periodic task s' that executes on a second processor P_2. However, tasks s and s' use a shared resource throughout their execution, and thus cannot simultaneously execute. This shared resource is not required by either of the feedback jobs. The access to the shared resource is fully preemptive with higher priority assigned to task s'. In other words, whenever a job of s' arrives, the shared resource is immediately handed over to s', and the job of s is blocked. Thus, we have a partitioned fixed priority scheduler with two processors and a shared resource.

A practical example of the above setting is in a big data application. Here task s periodically reads from a large database and generates certain operations, which are performed by the feedback jobs. In addition, periodically, the database is updated (written) by the task s' for shorter duration of times. During this update, the database cannot be read by s. However, while the database is being updated any backlogged feedback job can be performed.

B. Analytical Results

We first analyze the response time of the jobs of s based on the finish time of the previous job. Since there are two dependent jobs with a maximum delay greater than the period, the response time of a job of s can depend on the response times of two previous jobs.

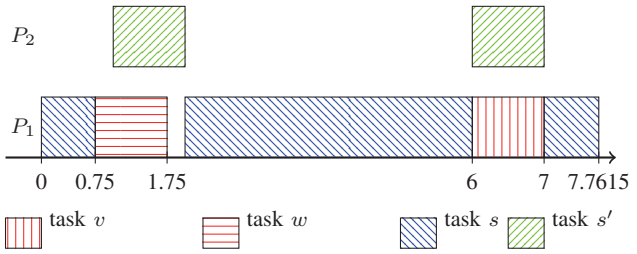

Fig. 13. Illustration of response times for the multiprocessor scheduler with $R_s(i-1) = R_s(i-2) = 7.75$.

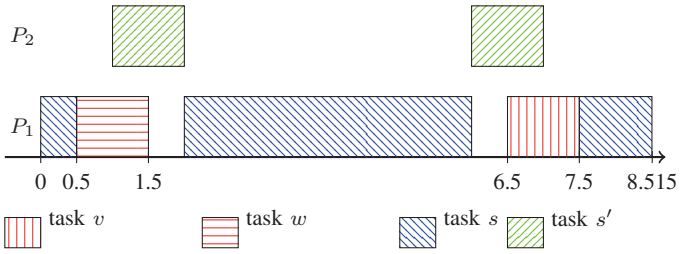

Fig. 14. Illustration of response times for the multiprocessor scheduler with $R_s(i-1) = 8.25$ and $R_s(i-2) = 7.5$.

Let the response time map be a two-dimensional map given as

$$R_s(i) = M(R_s(i-1), R_s(i-2))$$

A job of s is blocked by jobs of s' due to the shared resource. Since, in each period of s, there are two periods of s' both of which can interfere with the job of s, we have the blocking interference of $I_b = 2$. In addition, there is interference from the two dependent jobs, which we look at individually.

Consider a job of task v. If the response time of the $(i-1)$th job of s is $R_s(i-1) = 7.75$, then the corresponding job of v arrives at time $iP+6$. As shown in Fig. 13, this coincides with the blocking of the shared resource by s'. Hence, in this case the additional interference introduced by the job of v is 0. If $R_s(i-1)$ deviates from 7.75 on either direction, up to a maximum of 1, then the interference from the job of v increases linearly with slope 1. This is illustrated in Fig. 14, where $R_s(i-1) = 8.25$ and the interference from v increases to 0.5. In conclusion, the interference due to a job of v is given by $I_v = |R_s(i-1) - 7.75|$, for $R_s(i-1) \in [6.75, 8.75]$.

Consider the job of task w. If the response time of the $(i-2)$th job of s is $R_s(i-2) = 7.75$, then the corresponding job of w arrives at time $iP + 0.75$. As shown in Fig. 13, 0.75 time units of the execution of this task coincides with the blocking of the shared resource by s'. Thus, the additional interference is 0.25. If $R_s(i-2)$ is lower than 7.75, up to 7, then the interference increases linearly with slope 1. On the other hand, if $R_s(i-2)$ is higher than 7.75, up to 8, then the interference decreases linearly with slope 1. This is illustrated in Fig. 14, where $R_s(i-2) = 7.5$ and the interference from w increases to 0.5. Thus, the interference due to a job of w is given as $I_w = 0.25 - (R_s(i-2) - 7.75)$, for $R_s(i-2) \in [7, 8]$.

Considering the three components of interference, the response time of the ith job of s is given as

$$\begin{aligned} R_s(i) &= C + I_b + I_v + I_w \\ &= 5.515 + 2 + |R_s(i-1) - 7.75| + \\ &\quad (0.25 - (R_s(i-2) - 7.75)) \\ &= 7.765 + |R_s(i-1) - 7.75| - (R_s(i-2) - 7.75) \end{aligned} \quad (5)$$

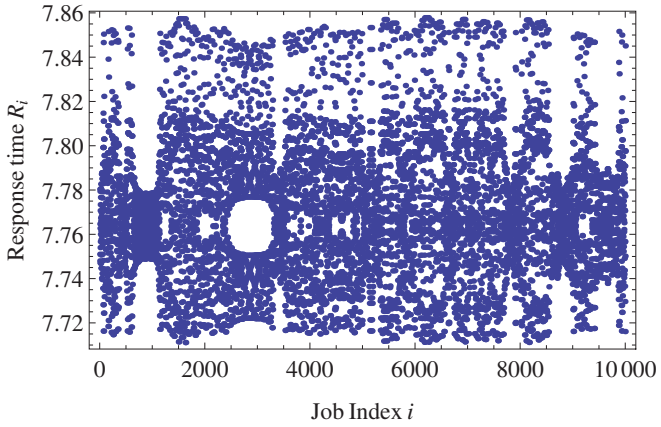

Fig. 15. Response time across jobs for the multi-resource scheduler with $R_s(i-1) = 7.76$ and $R_s(i-2) = 7.74$.

This maps holds under the condition that $R_s(i) \in [7, 8],\ \forall\ n > 0$.

C. Simulation Results

We numerically simulate the response times, using the above map, starting with $R_s(-1) = 7.76$ and $R_s(-2) = 7.74$. The corresponding response times are shown in Fig. 15. No apparent regularity is observed. The two-dimensional response time map is shown in Fig. 16.

The map of (5) is a scaled and shifted version of the gingerbreadman map [12] given as

$$x_n = 1 + |x_{n-1}| - x_{n-2}. \quad (6)$$

The gingerbreadman map is known to be chaotic [13] for any starting point within the dark shaded area of Fig. 16. In other words, the shown map is the map for all initial conditions which belong to the shaded area. Furthermore, for any initial condition starting in the six enclosed hexagons, the map is periodic with a period of 6, see [13].

D. Remarks

- The response time map of (5) is only a numerical example. Indeed, similar chaotic properties will hold for other values of the parameters, if suitably chosen.
- While the example of the round robin scheduler from the previous section was an example of a one-dimensional map that shows sign of chaotic behavior, the example from this section is a two-dimensional chaotic map.
- To the best of our knowledge, this is the first demonstration of the gingerbreadman map in any practical setting, more so with a standard scheduler algorithm.
- Unlike the map of Fig. 7 which showed signs of chaotic behaviour for all initial conditions, the map of (5) is chaotic for certain initial conditions.
- The existence of multiple processors is essential for the chaotic behavior to be observed. We see the required non-monotonic behavior due to the multiple processing elements: An earlier release of a job leads to a larger response time of a later job.

E. Implementation Results

To confirm the analytical results and to illustrate the occurrence of chaos in standard scheduling algorithms, we implemented the previously described setting on a standard computer system.

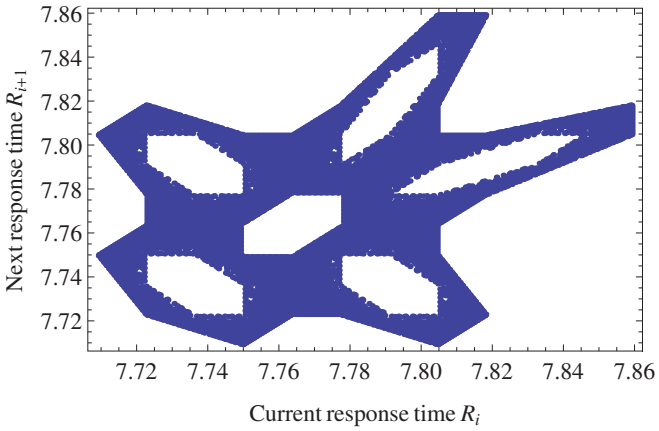

Fig. 16. Response time map for multi-resource scheduler with $R_s(i-1) = 7.76$ and $R_s(i-2) = 7.74$.

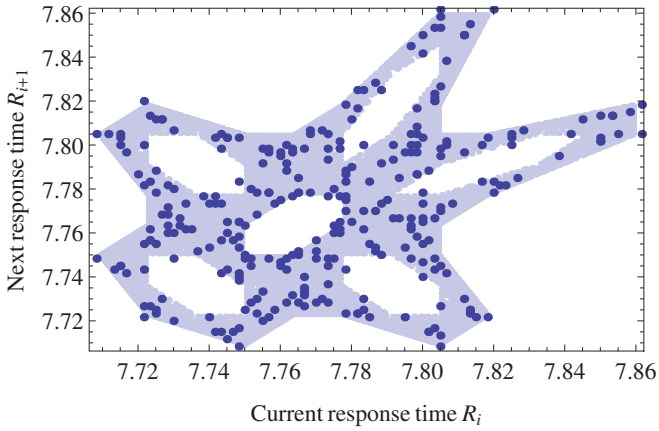

Fig. 17. Measured response time map for multi-resource scheduler implemented with `POSIX` threads on multiple cores.

We used separate `POSIX` threads to implement the tasks. Tasks s and s' run on different cores, but share a semaphore, which blocks s whenever s' is executing. The dependent jobs of tasks v and w are executed in separate threads which are invoked by programming hardware timers on the completion of a job of s. These feedback jobs run on the same core as s, but at a higher priority. The `SCHED_FIFO` scheduler, that is standard on a Unix scheduler, is used with different `RT` priorities assigned to the threads. This scheduling scheme is executed on an Intel Core i7 2.8 GHz processor running the Ubuntu operating system. As this is clearly not a real-time system, we chose timing parameters such that 1 time unit is equivalent to 1 s in real-time and therefore, scheduling overhead can be neglected for our purposes.

We execute the tasks according to the parameters specified in the setting and observe the finishing times of jobs of s. We choose an initial condition such that we start in the dark shaded area of Fig. 16. In this case, the response times obtained are shown as dark points in Fig. 17. The light shaded area is indicative of Fig. 16. Note that all points remain within the shaded area (within a certain tolerance due to the timing inaccuracies of a non real-time implementation), which confirms the analytical properties of the gingerbreadman map. Furthermore, different response times throughout the region are observed. This plot confirms that the scheduler, implemented according to standard practices, leads to chaotic behavior. To the best of our knowledge, this is the first such implementation which demonstrates chaotic behavior on a standard hardware platform for a standard scheduling algorithm.

VI. Priority Inheritance Protocol

In this section, we will give the last example of a scheduling algorithm which leads to chaotic behavior. We will consider the priority inheritance protocol [14] on a single processor system, which is commonly used to manage shared resources across multiple tasks.

A. Setting

The setting is depicted in Fig. 18. In addition to task s and its three dependent tasks u, v and w, we have two other periodic tasks s' and s''. A fixed priority scheduler is used with different priorities for each set of jobs, given as $s < u < w < s' < v < s''$. In addition, throughout their execution, tasks s, s' and s'' share a critical resource protected by a semaphore. We use the priority inheritance protocol [14] wherein a task with the ownership of a shared resource has the highest priority amongst all tasks blocked on that resource. Thus, if a job of s' arrives while s is blocking the resource, then the priority of s is elevated to that of s'.

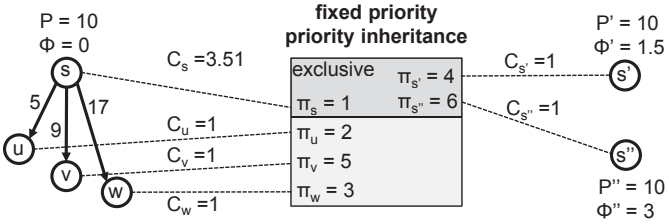

Fig. 18. Setting for the priority inheritance scenario.

B. Analytical Results

We begin with a response time analysis of the presented setting. Whenever the jobs of s and s'' arrive, the shared resource is held by s. Given the priority inheritance protocol, s suffers no interference from either of the tasks s' and s''. However, we need to consider the three dependent jobs which have nominal priority greater than that of s. We consider them separately. The maximum delay $(\max(D(s,u), D(s,v), D(s,w))$ is less than twice the period. Hence, the response time of a job of s would depend only on the response times of the earlier two jobs.

We analyze the response time of the ith job of s. Let $R_s(i-1) = 4$. Then, as shown in Fig. 19, the dependent job of u released by the $(i-1)$th job of s will cause no interference as it completes before the ith job of s is released. However, if $R_s(i-1)$ increases, the interference will increase linearly with slope 1. Instead, if $R_s(i-1)$ decreases, the interference will remain at 0. This is shown in Fig. 20. Thus, the first interference is given as $I_u = \max(0, R_s(i-1) - 4)$ for $R_s(i-1) \leq 5$.

For $R_s(i-1) = 4$, as shown in Fig. 19, a job of v caused by the $(i-1)$th job of s will arrive at time $i \cdot P + 3$. At this time a job of s'' arrives and elevates the priority of s to higher than that of the second feedback job. Thus, the interference is 0. However, if $R_s(i-1)$ decreases, the interference will increase linearly with slope 1. This is shown in Fig. 20. Instead, if $R_s(i-1)$ increases, the interference will remain at 0. Thus, the second interference is given as $I_v = \max(0, 4 - R_s(i-1))$ for $R_s(i-1) \geq 3$.

For $R_s(i-2) = 4$, as shown in Fig. 19, a job of w caused by the $(i-2)$th job of s will arrive at time $i \cdot P + 1$. It will execute

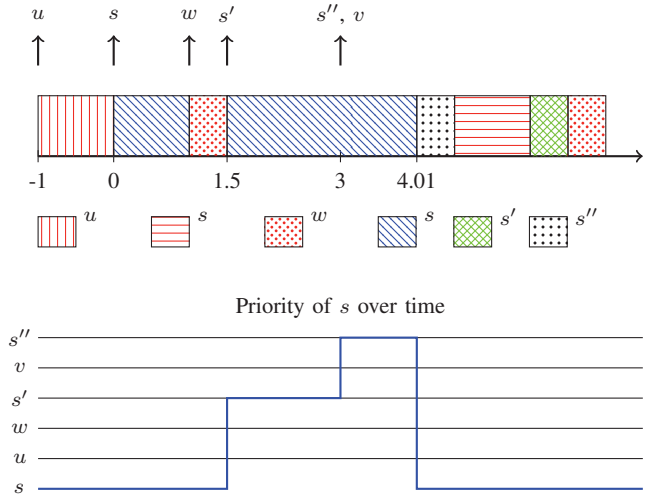

Fig. 19. Illustration of response times for the priority inheritance protocol with $R_s(i-1) = R_s(i-2) = 4$.

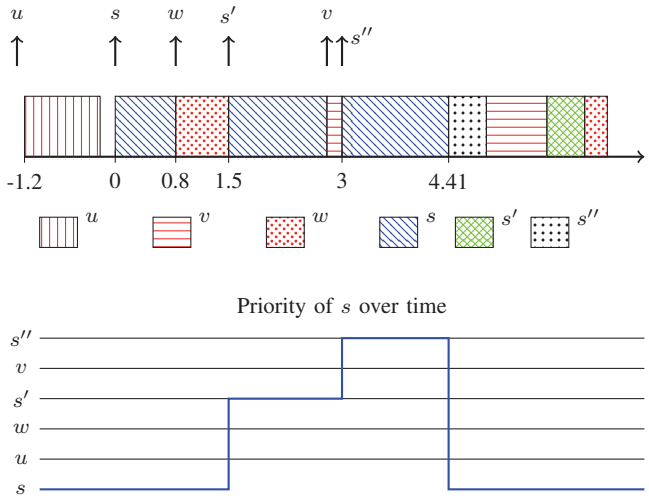

Fig. 20. Illustration of response times for the priority inheritance protocol with $R_s(i-1) = R_s(i-2) = 3.8$.

for 0.5 time units at which time a job of s' arrives and increases the priority of s to above that of the third feedback job. Thus, the interference is 0.5. As $R_s(i-2)$ increases from 4, the interference will decrease linearly with slope 1. On the other hand, as $R_s(i-2)$ decreases from 4, the interference will increase linearly with slope 1. This is shown in Fig. 20. Thus, the third interference is given as $I_w = 0.5 - (R_s(i-2) - 4)$ for $R_s(i-2) \in [3.5, 4.5]$.

Combing the above, the response time R_i is given as

$$\begin{aligned} R_s(i) &= C_s + I_u + I_v + I_w \\ &= 3.51 + \max(0, R_s(i-1) - 4) + \max(0, 4 - R_s(i-1)) \\ &\quad - (0.5 - (R_s(i-2) - 4)) \\ &= 4.01 + |R_s(i-1) - 4| - (R_s(i-2) - 4) \end{aligned} \tag{7}$$

This relation holds for $R_s(i) \in [3.5, 4.5]$. This map is also a scaled and shifted version of the gingerbreadman map (6). Thus, it also exhibits the chaotic properties as seen in the previous example. We remark that the setting described here is only a numerical example. Other values of the parameters, when suitably chosen, will also exhibit similar chaotic behavior.

VII. Implications of Complex Dynamic Behavior

A. Conditions

In the present paper, we investigated four scheduling systems which show a complex dynamic behavior. The round-robin, multiprocessor and priority inheritance scenarios have been shown and proven to exhibit complex response time evolutions. Whereas we described the underlying mechanisms in great detail, it may be useful to answer further questions like: Why have these four examples been chosen? What are the conditions that may lead to complex scheduling behavior? What is the role of the DAG task model used in the paper? The following discussion attempts to provide some answers.

Choice of examples: We have decided to choose examples with very different scheduling disciplines (single processor vs. two processors, round-robin scheduling vs. fixed priority scheduling, and two resource sharing methods) in order to explore various ways how complex response time behavior can occur. In each of these cases, we tried our best to construct as simple as possible scenarios that show interesting dynamic behavior.

Conditions for complex behavior: It has been observed in the real-time analysis of distributed systems, that in many cases the complexity to obtain tight bounds on the response time of tasks increases if (a) there are resource feedback cycles, i.e., the response time of a current job of a task influences the response time of future jobs of the same task, and if (b) there are timing anomalies in the investigated real-time system, i.e., a larger response time of a job leads to a smaller response time of some other job, see e.g., [15]. Interestingly enough, these conditions are exactly those that appear to be essential for generating complex response time evolutions.

Resource feedback and the DAG model: Response time maps as depicted in Figs. 7 and 9 clearly show that a later response time $R_s(i)$ of a job from a task s depends on earlier response times, e.g., $R_s(i-1)$. The same holds for the two-dimensional gingermanbread map as observed in the multiprocessor and priority inheritance examples: $R_s(i) = a + |R_s(i-2) + b| - R_s(i-1)$ for some parameters a and b. A simple mechanism to generate such dependencies is the use of dependent tasks, see Section II: Finishing of a job triggers the release of some other jobs that interfere with the next job release. This way, we provide means to construct the above mentioned resource feedback cycles. It can not be excluded that similar indirect interferences between subsequent jobs of a task can occur in other task systems as well.

Timing anomalies: Following the analysis of response times as provided in Sections IV-B and IV-C, one can see that the existence of timing anomalies is essential for complex dynamic behavior: If the map has always a non-negative slope, then from any initial (response time) value, only monotonically increasing or decreasing evolutions are possible.

In order to show this, let us suppose that the response time map M is non-decreasing, i.e., $x \geq y \Rightarrow M(x) \geq M(y)$. Suppose now that the response times at iterations i and $i-1$ satisfy $x_i \geq x_{i-1}$. Then one can conclude $x_i \geq x_{i-1} \Rightarrow M(x_i) \geq M(x_{i-1}) \Rightarrow x_{i+1} \geq x_i$, and in a similar way $x_i \leq x_{i-1} \Rightarrow M(x_i) \leq M(x_{i-1}) \Rightarrow x_{i+1} \leq x_i$. A similar necessary property holds in two-dimensional maps as well, see the gingermanbread map with $R_s(i) = a + |R_s(i-2) + b| - R_s(i-1)$.

Therefore, in order to reach a complex dynamic behavior of the response time evolution, timing anomalies are essential: A large response time at some iteration i leads to a smaller response time at some later iteration.

B. Opportunities

Identifying that certain real-time systems exhibit chaotic behavior has three major implications.

On the one hand, it shows the underlying complexity of the scheduling algorithms. For instance, certain scheduling algorithms may lead to fractal behavior (the characteristic of the evolution of response times exhibits a pattern that repeatedly displays at every scale of parameter variations) or have complex attractors (a set of states towards the system tends to evolve under variation of starting conditions). Classifying different behaviors of scheduling algorithms and identifying the underlying causes is not only an interesting research challenge. It can be expected that different dynamic behavior leads to different analysis methods. Partial answers have been provided in the present paper, see for example the analysis methods in Sections III, IV-B and IV-C.

Secondly, the analysis of a real-time system which a complex dynamic behavior can be more challenging than that of simple systems. Two common methods to obtain worst-case timing properties or estimations thereof are simulation and use of formal methods to compute fixed points. Either of these cases may not work well for chaotic systems. By definition a chaotic system will not have a fixed point and careful abstraction is required to obtain tighter bounds on fixed point calculations. On the other hand, when using simulation, we cannot derive the length of the simulation needed to obtain bounds within a certain confidence interval. Furthermore, with sensitive dependence on input conditions, the initial state can crucially and unexpectedly affect the result of such a simulation-based analysis. On the other hand, the explicit formulation of the possibly multi-dimensional response time map may lead to novel ways of computing upper bounds on the response time, see e.g., Sections IV-B and IV-C.

Finally, while designing real-time systems that are chaotic, we may need to employ techniques to reduce the un-characterised variability. For instance, in a control system rapidly varying response times may not lead to good performance. Shaping of jobs/messages could be one way to lower the complexity in such systems. As an example, consider the case of the gingerbreadman map shown in Fig. 17. If by delaying one or more jobs, we ensure than the response-times lie in one of the hexagonal regions, then all future response-times will remain within this hexagonal region. In other words, by nudging the system into an attractor of the response-time map we can avoid the chaotic behaviour.

The fresh look at real-time systems from a dynamical systems viewpoint may open up new opportunities. One example is the run-time control of non-linear systems, see e.g., [16], [17]. By means of feedback control the dynamic properties of the response time evolution could be changed towards a more favorable behavior, e.g., a smaller worst-case bound or a smaller variation of response times. In addition, in the context of the above mentioned research thread of controlling chaos, scenarios with stochastically changing parameters have been investigated, such as varying execution times of jobs. These research directions are beyond the scope of the present paper and left for future work.

VIII. Conclusions

The study of real-time systems has primarily focused on obtaining lower and upper bounds on the response times of jobs. In this work, we studied real-time systems from the perspective of dynamical systems, and aimed to characterise the evolution of response times. In certain scheduling settings, dependent jobs affect the response times of future jobs in unexpected ways. We studied four examples of such settings. In the case of a standard fixed priority scheduling, we showed that a periodic behavior with a large orbit can be seen. Then we investigated the case of round robin scheduling, where the response time map showed complex behavior. We also studied the properties of the corresponding iterated map for different scheduling parameters. Finally, we presented two examples that use multiprocessor scheduling and the priority inheritance protocol, which result in the well known gingerbreadman map.

We believe this is the first study that demonstrates that standard scheduling algorithms can exhibit chaotic behavior. The novel map of (4) is a novel system that shows signs of chaotic behavior as well as a bifurcation diagram with self-similar behavior (Fig. 10). Future investigations may deal with varying execution times of tasks, and use feedback control in order to favorably influence the evolution of response times in complex scheduling systems.

References

[1] M. Maggio, F. Terraneo, and A. Leva, "Task scheduling: a control-theoretical viewpoint for a general and flexible solution," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4, p. 76, 2014.

[2] T. Beaumariage and K. Kempf, "The nature and origin of chaos in manufacturing systems," in *Advanced Semiconductor Manufacturing Conference and Workshop. 1994. ASMC 94 Proceedings. IEEE/SEMI*, pp. 169–174, IEEE, 1994.

[3] J. J. Bartholdi, D. D. Eisenstein, and Y. F. Lim, "Deterministic chaos in a model of discrete manufacturing," *Naval Research Logistics (NRL)*, vol. 56, no. 4, pp. 293–299, 2009.

[4] J. Schmitz, D. Van Beek, and J. Rooda, "Chaos in discrete production systems?," *Journal of Manufacturing Systems*, vol. 21, no. 3, pp. 236–246, 2002.

[5] A. Erramilli and L. J. Forys, "Oscillations and chaos in a flow model of a switching system," *Selected Areas in Communications, IEEE Journal on*, vol. 9, no. 2, pp. 171–178, 1991.

[6] G. Feichtinger, C. H. Hommes, and W. Herold, "Chaos in a simple deterministic queueing system," *Zeitschrift für Operations Research*, vol. 40, no. 1, pp. 109–119, 1994.

[7] G.-X. Yu and P. Vakili, "Periodic and chaotic dynamics of a switched-server system under corridor policies," *Automatic Control, IEEE Transactions on*, vol. 41, no. 4, pp. 584–588, 1996.

[8] H. Tanaka and T. Ushio, "Analysis of aperiodic oscillations in a flow model of a switching system," *International Journal of Bifurcation and Chaos*, vol. 13, no. 04, pp. 981–994, 2003.

[9] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *33rd Real-Time Systems Symposium (RTSS)*, pp. 63–72, IEEE, 2012.

[10] W. R. Inc., "Mathematica edition: Version 8.0," 2010.

[11] S. Strogatz, "Nonlinear dynamics and chaos: with applications to physics, biology, chemistry and engineering," 2001.

[12] R. L. Devaney, "A piecewise linear model for the zones of instability of an area-preserving map," *Physica D: Nonlinear Phenomena*, vol. 10, no. 3, pp. 387–393, 1984.

[13] R. L. Devaney, "Fractal patterns arising in chaotic dynamical systems," in *The science of fractal images*, pp. 137–168, Springer, 1988.

[14] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *Computers, IEEE Transactions on*, vol. 39, no. 9, pp. 1175–1185, 1990.

[15] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. G. Harbour, "Influence of different abstractions on the performance analysis of distributed hard real-time systems," *Design Automation for Embedded Systems*, vol. 13, no. 1-2, pp. 27–49, 2009.

[16] E. Schöll and H. G. Schuster, *Handbook of chaos control*. John Wiley & Sons, 2008.

[17] S. Boccaletti, C. Grebogi, Y.-C. Lai, H. Mancini, and D. Maza, "The control of chaos: theory and applications," *Physics reports*, vol. 329, no. 3, pp. 103–197, 2000.