A Real Time Datacenter Monitor System

Giovani Ferreira[1], **Gustavo Coimbra**[1], **Eduardo San**[1]

[1]CEUB - Centro Universitário de Brasília
Caixa Postal 4488 – 70.904-970 – Brasília – DF – Brazil

Abstract. *Data center is a critical part to keep business running smoothly and continuously, needing to be a non-stopped service or have 24x7 availability. Assure the availability is a hard problem and requires a lot of effort and study, but keep the personnel updated about the conditions of the room is a good start and is crucial to avoid worst problems. This paper aims to proof that a real time system can solve the warning time problem of the data center, bringing reliability to measurement of the conditions of the room and ensuring that at any signal of danger, the responsible for the data center will be alerted.*

1. Introduction

A data center is a facility used to accommodate computer systems and associated components, such as telecommunications, network devices and storage systems. It usually includes redundant or backup power supplies, redundant data communications connections, air conditioning, fire suppression and security devices[Janpitak and Sathitwiriyawong 2011] [Hassan et al. 2013].

Because of technology compaction, the information technology (IT) industry has experienced a large decrease in the floor space required to achieve a constant quantity of computing and storage capability. However, power density and heat dissipation within the footprint of computer and telecommunications hardware have increased significantly. The heat dissipated in these systems is exhausted to the room, which must be maintained at acceptable temperatures for reliable operation of the equipment. Data center equipment may comprise several hundred to several thousand microprocessors. The cooling of computer and telecommunications equipment rooms has thus become a major challenge [Schmidt et al. 2005].

In this scenario of challenges, the availability of the data center must be preserved. It is a critical part to keep business running smoothly and continuously and a downtime can be costly resulting in production and business losses. To assure the availability, a critical element that must be present in any control system is a way to warn that something is going wrong inside the room. It is an importante task that must occur as quickly as possible, increasing the risk of loosing data or equipments at every second of delay. This paper focus in proof that is possible to use a real time system to monitor the basic conditions of the data center. This work is divided in 5 sections. The second section conceptualizes important elements. The third explains and present the RTDMS - Real Time Data Center Monitor System -, exposing its task model, proofing the scalability and describing the fault tolerance subsystem. The section four presents the results obtained in tests and experiments and the last talk about the conclusions retrieved from this work.

2. Related Concepts

This section aims to conceptualize important topics about real time systems for complete understanding of the data center monitor and the results achieved by this paper.

2.1. Time

It's important to understand what is time and it's related concepts. For this purpose, the definitions provided by [Kopetz 2011] will be considered:

- The flow of time is a directed time line that extends from the past into the future.
- A cut of the time line is called an instant.
- Any ideal occurrence that happens at an instant is called an event.
- The present point in time, now, is a very special event that separates the past from the future.
- An interval on the time line, called a duration, is defined by two events, the start event and the terminating event of the interval.
- The instant when a result must be produced is called a deadline.

In the context of deadlines, [Kopetz 2011] also provides a good definition, classifying them in three different ways:

1. Soft deadline: If a result has utility even after the deadline has passed,
2. Firm deadline: If a result does not matter after the deadline.
3. Hard deadline: If severe consequences could result if a firm deadline is missed.

2.2. Real-time systems

For [Kopetz 2011], a real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced. [Lichtenstein et al. 1985] states that a real-time system is a reactive system, that is, it must react to stimuli from its environment within time intervals dictated by its environment. According to [Stankovic and Ramamritham 1990], a real-time computer system can be classified in 2 different ways:

1. Static real-time system: where all deadlines can be guaranteed a priori.
2. Dynamic real-time system: large, complex, distributed, adaptive, contain many types of timing constraints, need to operate in a highly nondeterministic environment, and evolves over a long system lifetime.

The type of the deadline affects the real-time system. If the system must met at least one hard deadline, it's called a hard real-time computer system. If no hard deadline exists, then the system is called a soft real-time computer system. The design of a hard real-time system is fundamentally different from the design of a soft real-time system. While the first must sustain a guaranteed temporal behavior under all specified load and fault conditions, it is permissible for the second to miss a deadline occasionally [Kopetz 2011]. [Stankovic 1996] also points that hard real-time systems usually cause several consequences, even death, when missed an important deadline.

A computation of a real-time system can be viewed as a sequence of event occurrences. Informally, events represent things that happen in a system. An event occurrence defines a point in time in a computation at which a particular instance of an event happens [Chodrow et al. 1991].

2.3. Tasks

For [Stankovic 1996] tasks can be classified as:

- Periodic task: activated every T units. The deadline for each activated instance may be less than, equal to, or greater than the period T.
- Aperiodic task: activated at unpredictable times.
- Sporadic task: an aperiodic task with the additional constraint that there is a minimum interarrival time between task activations.

Real-time scheduling is the process of creating start and finish times for sets of tasks such that all timing, precedence, and resource constraints are met.

3. The RTDMS

The RTDMS, that stands for Real Time Datacenter Monitor System, is a real time system aiming to provide a reliable and solid way to measure and monitor the conditions of a datacenter. Using sensors for temperature, humidity and flood, the RTDMS is able to quickly discover adverse conditions in a datacenter and reduce the warning time, so the support team have more time to take the appropriate interventions and avoid worse situations.

The Rate Monotonic algorithm implementation used is the one provided by the NilRTOS[1] library, and the source code of the prototype is available at github[2].

3.1. Tasks Model

The RTDMS have the following tasks model:

Task Name	Period (ms)	Computation Time (ms)	Deadline Type
Alarm	Aperiodic	400	Hard
Check Flood	1500	30	Soft
Check Temperature	1500	820	Soft
Check Humidity	1500	20	Soft

- Alarm Task: Responsible to warn that something went wrong in datacenter. The deadline is hard. To assure that the deadline will be fulfilled, this task blocks the execution of the other tasks and monopolizes the hardware.
- Check Flood: This task uses a rain sersor to check if there is a water flood in datacenter. The deadline is soft because if it fails sometimes, it's not a problem. An undetected flood a second and half ago does not reach very high levels in this hiatus. If the tasks detects a failure in the sensors, it activates the alarm task to warn that something is wrong.
- Check Temperature: Responsible to collect the data from the temperature sensors and treat them to provide a realiable measure. The deadline is soft, the temperature will not change abruptly in one second and half. The validation of the temperatures measured can deal with a failure of one sensor (measuring a absurd temperature value or not measuring anything).

[1] https://github.com/greiman/NilRTOS-Arduino
[2] https://github.com/giovanifss/Datacenter-Monitor

- Check Humidity: Collect the data from humidity sensors and process it to provide a valid humidity measure. If the dealine fail sometimes, it's not a problem, humidity needs more time to significantly changes. This task can deal with failure of one sensor, correcting the measure and providing a correct final result.

3.1.1. Time Restrictions

The RTDMS time restriction is the warning time. In the moment that RTDMS detects an adverse condition, the alarm task must be execute instantly. The failure of this task, not warning or warning with delay, can cause big problems to the owners of the datacenter, including the loss of all data and equipments.

3.1.2. Scalability

The rate monotonic, which assigns the highest priority to the most frequent periodic task [Stankovic 1996], was considered an optimal fixed priority algorithm by [Liu and Layland 1973], which proved that for a set of n periodic tasks with unique periods, a feasible schedule that will always meet deadlines exists if the CPU utilization is below a specific bound (depending on the number of tasks). The schedulability test for Rate Monotonic Scheduler is:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1)$$

Where U is the CPU utilization, C_i is the computation time of the task i, P_i is the release period (with deadline one period after), and n is the number of processes to be scheduled. The set of tasks for RTDMS was mathematically proved as scalable:

1. Check Flood:
$$C_1 = \frac{C_1}{P_1} = \frac{30}{1500} = 0.0200$$
2. Check Temperature:
$$C_2 = \frac{C_2}{P_2} = \frac{820}{1500} = 0.5467$$
3. Check Humidity:
$$C_3 = \frac{C_3}{P_3} = \frac{20}{1500} = 0.0133$$

- Schedulability

$$U = \sum_{i=1}^{3} \frac{C_i}{P_i} = 0.7600 \leq 3(2^{\frac{1}{3}} - 1) = 0.7797$$

$$U = 0.7600 \leq 0.7797$$

3.2. Fault Tolerance

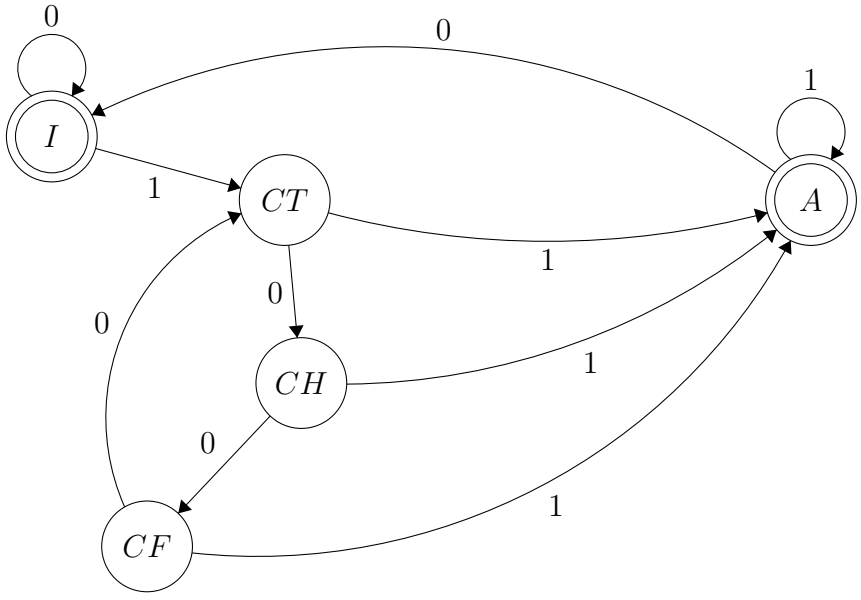
The RTDMS provides a tolerance for sensors fault, being able to deal with $\frac{1}{3}$ of sensors failing at the same time. The fault tolerance subsystem behave in the following way:

- Let $s_0, s_1, ..., s_n$ be sensors for either temperature or humidity. The system takes as premise that at least $50\% + 1$ sensors will measure the correct temperature;
- After compute the basic mean $\frac{s_0, s_1, ..., s_n}{n}$, the system checks for the absolute difference d for each sensor i from the mean: $d_i = |s_i - mean|$;
- Then, after check for values outside the margin of error acceptable, i.e. $d_i > acceptable\ value$, the system disregards measurements that does not respect the limit, these sensors disregards are represented by q;
- If a measurement is disregards, i.e. $q > 0$, the system recomputes the mean for the realiable values and outputs it as the final temperature: $\frac{s_x, s_y, ..., s_z}{n-q}$.

This behavior provides a tolerance for absence of sensor (disconnected or not working) and for fault in read (measuring an absurd value that is not aligned with the real temperature). Further, it still support the variance in read that sensors have when measuring, accepting values that are in the tolerable range of mistake.

3.3. System's Flow

The system follows a flow that aims to provide the maximum of autonomy to the task with hardest deadline, executing it with total hardware control to ensure that the deadline will be fulfilled. The flow can be seen in the following state machine:



- Initial State (I): Outputs '1' when the system is activated;
- Alarm Task (A): Outputs '0' when is turned off manually;
- Check Flood (CF): Outputs '1' when detects a considerable amount of water or a failure in the sensor;
- Check Temperature (CT): Outputs '1' when identifies a temperature above what it should be;
- Check Humidity (CH): Outputs '1' when the humidity is above the acceptable limit.

4. Experiments and Evaluation

The RTDMS was tested in several situations with different focus on the tests. The results for specific tests are shown below:

4.1. Task Set Schedulability

The task set for RTDMS was simulated in Cheddar scheduling simulator tool[3]. Being submitted to 15 tests, the result for all of them is: No deadline missed in the computed scheduling: the task set is schedulable if you computed the scheduling on the feasibility interval. The following pattern was achieved for all tests with no unforeseen problems:

- Check Temperature Task, with the highest priority, reaches the CPU at the beggining of the period, represented by t_0. After the time needed to execute the task (820ms), Check Temperature leaves the CPU at t_{820};
- At the point of time t_{820}, Check Humidity Task assumes the CPU. Computing for 20ms, leaves the processor at t_{840};
- Check Flood Task, with the lowest priority, get access to CPU at t_{840} and computes until t_{870} finishing the computation for the set of tasks.
- The CPU became inactive until the start of the next period, at t_{1500}.

This pattern was obtained in all executions of the test, even in the calculation of the worst case for all tasks, proving that this set of task is schedulable with a time gap (idle time) of 630ms.

4.2. Fault Tolerance

The RTDMS's fault tolerance subsystem proved reliability for fault of one of the three sensors. The check humidity and check temperature tasks were submitted to two different tests:

4.2.1. Sensor Absence

The Sensor Absence test consists of disconect or change the wire connections of a sensor in hardware, making the result obtained from a read from sensor be an absurd value or not be a number. The expected behavior of the system was to identify a strange conduct by the sensor and ignore its measure for this cicle of temperature or humidity check. Both tasks were submitted to this test three times, each of them with a different sensor being removed or altered. The subsystem's actions were logged and the results were uniform.

Firstly, the subsystem calculates the difference of the values acquired from the sensors from the mean. Then it detect that one sensor have a bigger difference than others. After check that this difference is beyond of the acceptable limit, it recomputes the mean for the sensors that presented a valid measure. The subsystem consider all sensors as susceptible of failure, bringing success in all tests, proofing that it's flexible to identify any sensor fault.

On the other hand, if two or more sensors were compromised, the system fails to detect the correct temperature or humidity and outputs a invalid value. This limit can be extended adding more sensors because the way that the subsystem checks for error have the property that the more sensors, more reliable is the system.

[3]http://beru.univ-brest.fr/ singhoff/cheddar/

4.2.2. Controlled Sensor

This test main goal is to see whether the system is able to detect a valid temperature measure but incorrect. It consists of manipulating the sensor measure to valid but incorrect values and monitor the system behavior step by step. The test was repeated three times, each of them with a different sensor being manipulated. The subsystem had the same behavior as the previous test, detecting an fault by computing the difference from the mean.

The subsystem presented a success for all repetitions of the test, being able to deal with one sensor failure in each iteration. And as expected, the subsystem fails to provide a reliable measure when two or more sensors present an misleading measure.

5. Conclusion

This paper proves that a real time system can be used to monitor the basic conditions of a data center and reduce the warning time of any danger inside the room. Consisting of a set of tasks mathematically schedulable - and reaching success in scheduling simulator tests - alongside of a fault tolerance subsystem, the system is able to check for basic adversities in a realiable way, solid and efficient. On the other hand, a data center monitor system to real world implementation must be aware of more conditions beyond temperature, humidity and water flood. A complete system must provide treatment for airflow heat, fire control and power management, being able to operate in different energy consumption policies.

Although this paper proves that a real time system can be used for monitor the basic conditions of a data center, more work is needed to develop a full system able to deal with all potential hazards. These hazards are outside of the scope of this paper and are a good start point for future works in this topic.

Referências

Chodrow, S. E., Jahanian, F., and Donner, M. (1991). Run-time monitoring of real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 74–83. IEEE.

Hassan, N., Khan, M. M. K., and Rasul, M. (2013). Temperature monitoring and cfd analysis of data centre. *Procedia Engineering*, 56:551–559.

Janpitak, N. and Sathitwiriyawong, C. (2011). Data center physical security ontology for automated evaluation. In *The 2011 International Conference on Security and Management (SAM2011)*, pages 78–84.

Kopetz, H. (2011). *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media.

Lichtenstein, O., Pnueli, A., and Zuck, L. (1985). *The glory of the past*. Springer.

Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61.

Schmidt, R. R., Cruz, E., and Iyengar, M. K. (2005). Challenges of data center thermal management. *IBM Journal of Research and Development*, 49(4.5):709–723.

Stankovic, J. A. (1996). Real-time and embedded systems. *ACM Computing Surveys (CSUR)*, 28(1):205–208.

Stankovic, J. A. and Ramamritham, K. (1990). What is predictability for real-time systems? *Real-Time Systems*, 2(4):247–254.