Adding support to the ALMA Common Software for Real-Time operations through the usage of a POSIX-compliant RTOS

Rodrigo J. Tobar[a], Mauricio A. Araya[b], Tomas Juerges[c] and Horst H. von Brand[a]

[a]Computer Systems Research Group, Universidad Técnica Federico Santa María (UTFSM), Valparaíso, Chile

[b]Institut National de Recherche en Informatique et Automatique (INRIA), Nancy, France

[c]National Radio Astronomy Observatory (NRAO), Socorro, NM, USA

ABSTRACT

The ALMA Common Software (ACS) framework lacks of the real-time capabilities to control the antennas' instrumentation — as has been probed by previous works —, which has lead to non-portable workarounds to the problem. Indeed, the time service used in ACS, based in the Container/Component model, presents plenty of results that confirm this statement. This work addresses the problem of design and integrate a real-time service for ACS, providing to the framework an implementation such that the control operations over the different instruments could be done within real-time constraints. This implementation is compared with the current time service, showing the difference between the two systems when subjecting them to common scenarios. Also, the new implementation is done following the POSIX specification, ensuring interoperability and portability through different operating systems.

Problem

The ALMA Common Software (ACS) [1] is the common distributed control framework used throughout the entire ALMA project [2]. This framework provides common patterns and facilities for software development and distributed control, but does not offer hard real-time capabilities to the users.

This problem arises, for example, in the ACS Time Service. This framework subsystem is based in the Container/Component model [3,4], with a `Timer` component (CORBA Servant) in charge of triggering the executions of periodic tasks. This system has been proved to be non hard real-time [5]. In the other hand, the real-time requirements of the ALMA project have been solved by means of solutions outside the scope of ACS: special kernel-space modules are in charge of dealing with real-time operations, making them difficult to maintain and port to other systems. These features, instead, should be offered by the common framework. Our work presents a new implementation of the ACS Time Service using the Linux RT_PREEMPT kernel, which aims to solve this problem.

Linux RT_PREEMPT

One of the newest solutions in the world of the real-time operating systems (RTOS) is the usage of the Linux kernel with the RT_PREEMPT patch [6]. This modification to the Linux kernel adds real-time capabilities to the operating systems, making it suitable for hard real-time operations, even from the user-space. Its usage is currently growing up, at the point that its usage has been studied within the ALMA project [7]. To test the real-time capabilities of the patch, we stressed the test machine while running the `cyclictest` [8] with the following tasks:

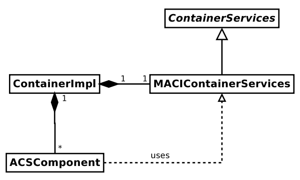
	Kernel	RT_PREEMPT			*vanilla*		
	Load avg. (max)	4.67(5.17)			4.43(4.57)		
Interval	Priority	Min	Avg	Max	Min	Avg	Max
100	60	13	27	**87**	1	23	614
600	59	19	32	**83**	11	34	585
1100	58	19	28	**118**	13	26	523
1600	57	14	30	**104**	12	27	721
2100	56	17	34	**187**	12	30	789
2600	55	30	42	**166**	12	28	516

- `scp`-ed a DVD iso image through the Ethernet interface,
- Copied once and again a big file to a new one.
- Compiled a C++ application, cleaned it, and compiled it again.
- Ran the compiled application (Genetic Algorithm)

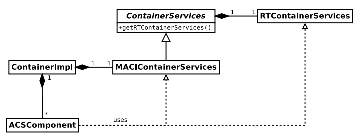
The test longed 60 seconds, with 6 real-time threads running simultaneously. The table at the left summarises the results for this setup, showing the statistics for the differences between the theoretical and experimental values of the realization of the real-time tasks (in [μs]) with a base interval of 100 [μs].

With the obtained results, it is clear that the Linux RT_PREEMPT-patched kernel is superior that the original one regarding to the determinism on the timing of its operations, on at least one order of magnitude.

Solution

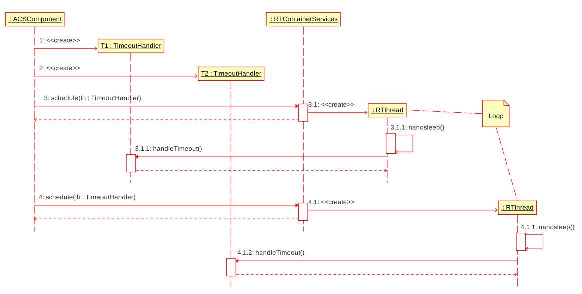


The proposed solution is based in the Container/Component model. This model encourages to embed all the services that a Component may use into the Container. To the current design of the C++ implementation of the Component/Container model (figure on the left) we added a new class that is used by the Component to call real-time-related methods (figure to the right).



The main method that was intended to be implemented was the `schedule` method, defined in an IDL interface in ACS. We kept the interface as similar as possible to the original one, thus providing an easy-to-migrate-to solution. Matter of fact, the class that implements the periodic task does not need changes at all.

To run a periodic real-time task, the user must register it through the `RTContainerServices` class. Each task will trigger the creation of a new thread with real-time priority. Once created, the following steps will be taken:



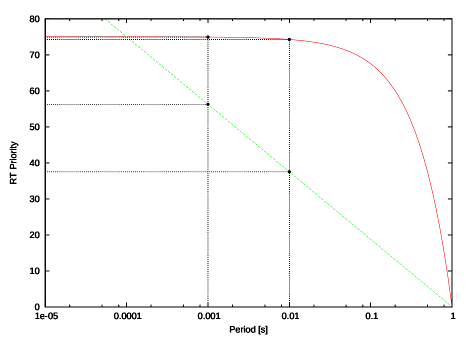
- Optionally, the thread will wait for a fixed amount of time to start the execution of the periodic task.
- Then, it calculates the next instant when it should execute the task and sleeps until then using the nanosleep POSIX call. This call features nanosecond precision, thanks to the high-precision timers implemented by the Linux RT_PREEMPT kernel.
- After sleeping, the thread will execute the task, calculating the start and end timestamps with the clock_gettime POSIX call.
- Finally, it will repeat the previous steps until canceled.

To schedule the different jobs through time we use the *rate monotonic* algorithm [9]. To assign a priority to each task, we use a logarithmic function, which depends on the period of the given task, and with adjustable maximum and minimum values for both period and priority values (see equation 1). This logarithmic relation handles gracefully the different orders of magnitude that the period values can present, and at the same time deals better with the different granularities of the two quantities (period values are continuous, while the priorities range from 0 to 100).

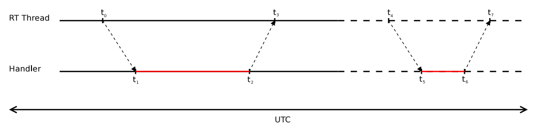
$$P_{\log}(T) = P_{min} - \left(\frac{P_{max}-P_{min}}{\log_{10}(T_{min})-\log_{10}(T_{max})}\right)\cdot T_{max} + \left(\frac{P_{max}-P_{min}}{\log_{10}(T_{min})-\log_{10}(T_{max})}\right)\cdot \log_{10}(T) \quad (1)$$

For our particular case, we define the minimum and maximum values as follows: $P_{max} = 75$, $P_{min} = 0$, $T_{max} = 1$[s] and $T_{min} = 100$[μs]. These values are in agreement with the needs of the ALMA project, and, in general, are expected to value for a great variety of requirements.

Finally, the red line in the figure represents equation 1 with the parameters mentioned above. The green line presents what would be a simple linear assignment with the same border conditions.



Experiments



The experiment setup used to test our new implementation considers four timestamps that repeat on each execution of the periodic real-time task, as shown in the figure on the left. Independently of the realization of a periodic task execution, we consider the task as if it would be absolute periodic (θ) or relative periodic (ϕ) [5]. This difference arises from the nature of each requirement: the time interval from execution to execution of a task can be driven by an initial instant on time (t_0), or by the previous execution instant of the task ($t_{4(i-1)}$). Since t_{4i} represents the start of the execution of the real-time task in the scheduling thread, we consider this instant as our main reference.

For a realization of the real-time task with N executions and a period P, we collect the measured differences for the task when considered as an absolute periodic requirement (Δ_{abs}, equation 2) and when considered as relative periodic (Δ_{rel}, equation 3).

$$\Delta_{abs} = \{\delta^0_{abs}, \delta^1_{abs}, \cdots, \delta^{N-1}_{abs}\} \qquad \delta^i_{abs} = i \cdot P - (t_{4i} - t_0) \quad (2)$$

$$\Delta_{rel} = \{\delta^0_{rel}, \delta^1_{rel}, \cdots, \delta^{N-1}_{rel}\} \qquad \delta^i_{rel} = \begin{cases} 0 & \text{if } i = 0 \\ P - (t_{4i} - t_{4(i-1)}) & \text{otherwise} \end{cases} \quad (3)$$

Over these two set of data (Δ_{abs} and Δ_{rel}) we calculate the maximum between the absolute value of the differences (*jitter*), and the root square of the Mean Square Error (MSE) of the values as a measurement of the concentration of the differences around the mean value. We use two different deployments to check the Time Services:

- **One client**: We register only one real-time task. This deployment measures the behavior of the system without stress.
- **Harmonic clients**: We register six real-time tasks, the first with period P, the next with period $2P$ and so on. This stresses the system to meet several deadlines at the same time, and therefore introduces more difficulty to the problem.

For each case we used base periods of 4 [ms], 24 [ms] and 48 [ms], simulating the real-time requirements of the CONTROL subsystem in the ALMA project [10]. The computer used for the experiments was a Pentium III (Coppermine) @644 Mhz with Arch Linux, and a `2.6.26.3-rt5` kernel.

Results

These first results for a single client setup show a high contrast between the two time services, and even worse results for the ACS Time Service than those reported by [5], confirming the fact that the current ACS Time Service does not offer real-time capabilities. In all cases our implementation had a better jitter, with differences up to 2 orders of magnitude (in ϕ_{24}) with the jitter presented by the ACS Time Service implementation. The differences are even more notorious for the harmonic clients case, where there are differences of up to 4 orders of magnitude (in θ_4). The worst case executions for each implementation have an error near to 30 000 % for the ACS Time Service (θ_4, harmonic clients deployment), and of 8 % for the our implementation (ϕ_4, harmonic clients deployment).

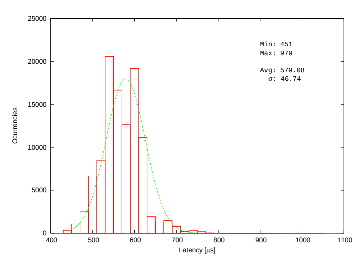
Type	One client				Harmonic clients			
	Jitter [μs]		$\sqrt{\text{MSE}}$ [μs]		Jitter [μs]		$\sqrt{\text{MSE}}$ [μs]	
	ACS	RT	ACS	RT	ACS	RT	ACS	RT
θ_{48}	89	**75**	22.90	19.36	6127	**60**	1527.17	42.87
θ_{24}	2309	**76**	34.05	44.82	4017	**58**	1514.2	39.47
θ_4	1453	**173**	20.83	26.01	1247292	**291**	724972.34	35.46
ϕ_{48}	102	**92**	5.54	3.59	6142	**66**	2172.78	5.69
ϕ_{24}	2335	**94**	38.51	2.97	4032	**80**	2157.17	5.13
ϕ_4	1278	**193**	19.21	4.28	11555	**321**	1978.72	5.02

After the first experiments, and in order to diminish the possible causes of jitter, we tuned the `ext3` partitions of the machines to be mounted in asynchronous mode, with a commit interval greater than a single experiment's duration. This way, we avoid all disk transfers, maintaining the information in the filesystem cache in RAM, in effect simulating a diskless computer during the execution of the experiments, and thus we have a less IRQ-ed system.

Type	One client				Harmonic clients			
	Jitter [μs]		$\sqrt{\text{MSE}}$ [μs]		Jitter [μs]		$\sqrt{\text{MSE}}$ [μs]	
	ACS	RT	ACS	RT	ACS	RT	ACS	RT
θ_{48}	**62**	67	17.67	27.47	1977	**49**	1159.96	35.2
θ_{24}	2036	**80**	31.96	32.11	1974	**46**	1133.08	30.92
θ_4	749	**47**	24.01	20.73	842251	**75**	465753.11	37.41
ϕ_{48}	81	**13**	5.41	2.49	2001	**39**	1654.58	4.99
ϕ_{24}	2052	**14**	36.95	1.86	1999	**41**	1616.53	6.57
ϕ_4	804	**59**	14.17	2.05	11945	**36**	1477.07	3.07

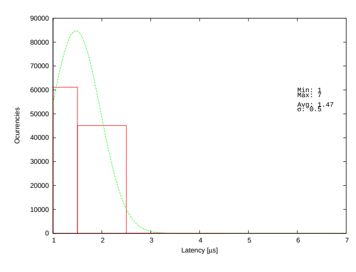
In these experiments, only the θ_{48} requirement (in the one client deployment) was (just a little) better served by the current ACS Time Service, while the rest were totally dominated by our implementation. Once again, the harmonic clients setup showed the worst results for the ACS Time Service, while our implementation stayed with low jitter, which even got reduced in some cases. Most important, all the jitter values of our implementation are well below the 150 [μs], so we can claim that this Time Service does provide real-time support to ACS, and complies with the ALMA real-time requirements.

Latencies



We also studied the latencies that the system experimented. The latencies are defined as the differences between the instant when the call is made from the scheduling thread and the instant when the execution really start ($t_{4i-1} - t_{4i}$).

The figure on the left shows the latencies for the current Time Service implementation. A high mean (about 580 [μs]) and standard deviation put into evidence the low reliability of this implementation. These results are due to the huge CORBA call stack that is produced between the scheduling component and the action handler.



In contrast, the figure on the right shows the latencies for the new implementation. Since this new implementation uses direct calls to the action handler, the latencies are highly minimized, falling to no more than 7 [μs] in the worst case.

Conclusions

The Linux RT_PREEMPT kernel shows a much more predictable behavior than the common *vanilla* Linux kernel, even when the machine is put under the stress of heavy disk operations, network traffic and CPU-intensive applications. This makes it suitable for hard real-time tasks, such as automatic control systems. Linux RT_PREEMPT has proved to be a good new approach among the open source real-time kernel because its design, keeping its monolithic form, but offering proven real-time performance on several architectures. This opens a large number of new possibilities for the Linux world to be present in the control industry. The current large activity around the development of this patch is reflected on a quick release cycle, good technical assistance through mailing list and forums, and a growing community, supported by the already existent Linux kernel development community.

Also, the new implementation of the Time Service does provide real-time support for the ALMA Common Software, at least at the level of the requirements of the CONTROL subsystem. Thanks to its design and the usage of a POSIX specification, our new implementation presents much lower jitter values when stressing the system with several requests. Even more, in low stressed scenarios it also showed better behavior than the original one. This proves that ACS could support real-time operations from user-space, and get rid of the kernel modules work-around solutions that are currently in use in the CONTROL and CORR subsystems.

Future Work

As a future work we would like to extend the study of our new implementation over other POSIX-compliant RTOSs. Since this work was only focused on the usage of Linux RT_PREEMPT, other RTOSs were left behind. QNX, for instance, would be a good candidate to be studied, but effort should first be spent in order to support QNX for ACS. Previous work has been done in this matter by the ALMA-UTFSM Group, but with not very inspiring results.

By testing our new implementation over other RTOSs would give us the opportunity to find even better results that those reported in this work, thus offering different alternatives to the ALMA Common Software real-time support, and extending its capabilities towards being a generic distributed control framework, reusable in any other project.

Acknowledgements

This work could not have been done without the support of the ALMA-CONICYT grant # 31060008. Rodrigo Tobar's work was also partially financed by the *"Programa de Iniciación en la Investigación Científica"* grant. Part of this trip was also financed by the *"Dirección General de Investigación y Postgrado"* of the UTFSM. Jorge Ibsen and Flavio Gutiérrez have been of great support during the entire work.

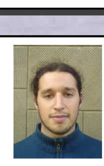
References

[1] Chiozzi, G. et al., "The ALMA Common Software, ACS status and developments," in [*Proceedings of ICALEPCS 2005*], (2005).
[2] Tarenghi, M., "The Atacama Large Millimeter/submillimeter Array: Overview & status," *Astrophysics and Space Science* **313**, 1–7 (Jan. 2008).
[3] Sommer, H. and Chiozzi, G., "Transparent XML binding using the ALMA Common Software (ACS) container/component framework," in [*Proceedings of the 13th Astronomical Data Analysis Software & Systems Conference*], (2003).
[4] Sommer, H., Chiozzi, G., Zagar, K., and Voelter, M., "Container-component model and XML in ALMA ACS," *Advanced Software, Control, and Communication Systems for Astronomy* **5496**(1), 219–229, SPIE (2004).
[5] Araya, M. A., *Verifying Real-Time Periodic Properties on the ALMA Common Software Time System*, Master's thesis, Departamento de Informática, Universidad Técnica Federico Santa María (May 2008).
[6] Rostedt, S. and Hart, D. V., "Internals of the RT patch," in [*Proceedings of the Linux Symposium 2007*], (2007).
[7] Pisano, J., Amestica, R., Juerges, T., and Jeram, B., "Real-Time Linux Migration Proposal," (Mar. 2009).
[8] Gleixner, T., "Cyclictest." `http://rt.wiki.kernel.org/index.php/Cyclictest`.
[9] Liu, C. L. and Layland, J. W., "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM* **20**(1), 46–61 (1973).
[10] Farris, A., Marson, R., and Kern, J., "The ALMA telescope control system," in [*Proceedings of ICALEPCS 2005*], (Oct. 2005).

NAOJ National Astronomical Observatory of Japan

NRAO

Atacama Large Millimeter Array



Contact e-mail: rtobar@csrg.inf.utfsm.cl / For more information about ALMA development and research at UTFSM Computer Systems Research Group, please visit our web site http://alma.inf.utfsm.cl