Distributed Real-Time Computing with Harness*

Emanuele Di Saverio[1], Marco Cesati[1], Christian Di Biagio[2], Guido Pennella[2], and Christian Engelmann[3]

[1] Department of Computer Science, Systems, and Industrial Engineering, University of Rome "Tor Vergata", Rome, Italy
[2] Applied Research & Technology Department, MBDA Italia SPA, Rome, Italy
[3] Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA
emanuele.disaverio@alice.it, cesati@uniroma2.it, {christian.di-biagio,guido.pennella}@mbda.it, engelmannc@ornl.gov

Abstract. Modern parallel and distributed computing solutions are often built onto a "middleware" software layer providing a higher and common level of service between computational nodes. Harness is an adaptable, plugin-based middleware framework for parallel and distributed computing. This paper reports recent research and development results of using Harness for real-time distributed computing applications in the context of an industrial environment with the needs to perform several safety critical tasks. The presented work exploits the modular architecture of Harness in conjunction with a lightweight threaded implementation to resolve several real-time issues by adding three new Harness plug-ins to provide a prioritized lightweight execution environment, low latency communication facilities, and local timestamped event logging.

Key words: Distributed Computing, Middleware, Real-Time, Harness, Plugin

1 Introduction

Parallel and distributed computing solutions provide the means for computational performance for High-End Computing (HEC) applications beyond the limits of single processor technology. The actual implementation of complex parallel and distributed software systems can be enormously eased by the adoption of an intermediate software layer, a "middleware". A middleware is defined as "... a connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network." [1]. A very specific topic of HEC applications is real-time computation.

* The research at Oak Ridge National Laboratory (ORNL) is sponsored by the Office of Advanced Scientific Computing Research; U.S. Department of Energy. ORNL is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

The term *real-time* pertains to computer applications whose correctness depends not only on results, but also on the time at which results are delivered. A *real-time system* (RTS) is a computer system that is able to run real-time applications and fulfill their requirements in a deterministic fashion. Thus, when defining a real-time system, we actually define requirements about its response time, meaning with this average value or the tail distribution of it. Distributed real-time systems development requires a well suited real-time oriented middleware as a supporting layer.

2 Previous Work

Traditional solutions in distributed real-time environments refer to Parallel Virtual Machine (PVM) [2] and Message Passing Interface (MPI) [3] libraries. Although PVM is a solid and simple solution, its process-based architecture is a little outdated, and the service set does not fit well into an industrial context. MPI is not well suited when compared to modern middlewares because it provides just a communication abstraction. More recent and rich products include Real-Time Innovations Data Distribution Service [4] (RTI DDS, formerly NDDS) and the Adaptive Communication Environment (ACE) Object Request Broker (ORB) in conjunction with TAO [5]. RTI DDS provides a communication abstraction data-centric layer that realizes a publish-subscribe semantic. It is a performing and well-featured product, which offers real-time oriented features, like a fine tunable QoS performance level, and an efficient low-latency implementation based on an open standard from OMG group [6]. However, it is especially suited for dynamically changing network topologies and to cover reliability issues. Moreover, it is a commercial (and thus closed) product, while in the industrial context being able to lower costs and customize the product at will is of key importance. The ACE ORB, on the other hand, is an open source project that implements the Object Request Broker semantic. TAO is a very complex and complete middleware, but it is designed around the ORB specification and is meant to be used with the existing plethora of CORBA services. This means that the internal architecture of TAO involves many different components. Moreover, its service-oriented nature makes it not easily tailorable for embedded applications. The approach described in this paper is more simple and streamlined, it avoids the role of the broker node, and it integrates nicely with the Harness framework.

3 Modern Middleware: Harness

Our effort focuses on an emerging technology in this field, the Harness project, a joint development effort between the Oak Ridge National Laboratory (ORNL), the University of Tennessee, and Emory University. Harness is a distributed, reconfigurable and heterogeneous computing environment that supports dynamically adaptable parallel and distributed applications. The unique feature of Harness relies on its almost total level of pluggability. The aim is to build a virtual

environment that can dynamically change (almost) anything at runtime. In this highly adaptable framework, several parallel and distributed user applications can reside, all executed on top of its distributed virtual machine (DVM) and runtime environment (RTE) concepts, a PVM successor. Harness runs a RTE on every computational unit as the "shell" in which it hosts the user applications and the resource management routines that belong to the distributed environment. Every RTE is realized by a Harness kernel, the core of the unit, which is capable of loading and unloading plugin modules and consists of a communication module, a module dedicated to process control, and a module dedicated to resource management, plus possibly a number of other plugins.

Several Harness prototypes have been developed, in Java and C. The work presented in this paper focuses on the C variant developed at ORNL [7], which runs on GNU/Linux. Its design focuses on a lightweight and pluggable middleware layer with a Harness kernel running as a Linux daemon process. The kernel performs process management, thread pool management, and dynamic plugin loading/unloading. The process management module can fork and execute a user application, and provides means for passing arguments, sending input, and retrieving output for these external processes. The thread pool is the heart of the lightweight execution environment provided by Harness. It creates a set of working threads that keep trying to empty a job queue data structure. It provides interfaces for adding a new job to the queue with proper arguments and cleanup function in case of thread cancellation. The plugin loader builds on top of the Linux dynamic library loader, and provides interfaces for loading/unloading modules and publishing their functionalities to the whole runtime environment. The communication facilities are effectively provided through RMIX (Remote Method Interface eXtensible) [8]. RMIX is a dynamic, heterogeneous, reconfigurable communication framework that allows software components to communicate using various RMI/RPC protocols by employing provider plugins in order to support different protocol stacks. RMIX emulates the Java RMI structure, allowing components to remotely call methods, retrieve the output, and export locally available methods.

4 Existing Real-Time Issues

Within the middleware definition stated in Section 1, we can outline a set of requirements expected from such a software. The first assumption we make is that the services on top of which the middleware is built retain themselves real-time capabilities. We cannot avoid this assumption because the aim is to work on a middleware that is a relatively high-level software, thus built on top of a set of services, and the performances of the resulting system are bound to those of that services. Secondly, remote services have to be designed without concurrency issues. In this context, the meaning of concurrency is twofold:

– **Call concurrency:** simultaneous access of the same service offered by the same computational resource

– **Service concurrency:** simultaneous access of different services offered by the same computational resource

A real-time middleware has to guarantee both kinds of concurrency support in order to be seamlessly scalable with the growth of the execution flows. Finally, services have to be locally executed without scheduling issues by providing a proper priority-aware execution environment. The hosted application should be able to run a number of real-time tasks that are expected to be scheduled in a deterministic manner. In order to address all these issues, three Harness plugins have been developed:

– A plugin to provide a prioritized lightweight execution environment
– A plugin for low latency communication facilities
– A plugin to support local timestamped event logging

5 Developed Plugins

The aim of our work is exploiting the pluggable nature of Harness in order to implement a set of services enabling the development and execution of successful real-time applications.

5.1 Real-Time Thread Pools

The lightweight execution environment provided by Harness is designed for great efficiency, but lacks in direct support for operating system scheduler directives. The thread pool solution is indeed a lightweight solution for job processing, but lacks the ability to exploit the preemptability of the latest Linux kernels. POSIX threads, like processes, can control their scheduling priority and contention scope, which can be set to either process or system scope. By using the latter it is possible to grant absolute schedule priority to the thread. Therefore, the first developed plugin focuses on providing a greater level of control on thread pools to user applications. It allows to define an arbitrary number of job classes, and for each level to specify the scheduler policy and priority of the related thread pool. Each pool tries to empty a different job queue. If not otherwise specified, the plugin is configured by default to create three pools of threads, in addition to the original one, it adds two pools entirely made of threads with real-time scheduling properties.

The first additional thread pool adopts a round-robin scheduling algorithm with priority p_1. The second additional thread pool uses a first-in/first-out scheduling algorithm with priority p_2, where $p_1 < p_2$ and global contention scope hold. This way we scaled the priority of the executing threads from one to three levels, allowing the application executing on top of Harness to have control over the scheduling of its tasks. The default configuration should provide more than enough means for executing real-time applications without worries for scheduling issues. If a more fine-grained solution is needed, it is still possible to explicitly specify the pool parameters.

5.2 Real-Time Remote Procedure Calls

The second plugin faces a problem of the RMIX framework in its actual form. The standard provider plugin builds on top of the TCP transport layer. This solution, while providing a reliable and stream oriented communication, is not well suited for distributed real-time applications. In order to address this issue, an RMIX provider plugin was built to implement the RMIX primitives over UDP, while keeping a real-time job (as explained in the previous subsection) serving the incoming and outgoing communications. This way we bring into the Harness middleware layer a more efficient implementation of the Remote Procedure Call communication scheme built onto a connectionless transport level. While losing the reliability and the complex acknowledgment system of TCP, we gain performance in response time metric, both in its absolute value and in variance of its distribution. This solution exploits the pluggable nature of the RMIX Framework, that itself is seen as a plugin by the Harness Framework. This double-layered pluggability has the added benefit of not requiring modifications to applications already using the RMIX Communication Framework.

5.3 Real-Time Event Logging

A common source of unpredictable latency is the access to file or screen I/O devices, as a plain `printf()` function call is a very time consuming task. While this effect can easily be ignored in standard distributed applications, it cannot be tolerated in a distributed real-time system. The third developed plugin implements a simple event logging system. Loading this module enables the application to push the event to be logged in a temporary shared buffer, while storing information about the source of the event, the timestamp, and the description of the event itself. This operation is a low overhead one, while the buffer will be emptied by a regular Harness thread waiting on the event queue, and optionally formatting the output in a simple XML or plain text file. This way an application can effectively maintain logs of events with accurate timestamps in a lightweight fashion, that is, without perturbating the execution environment as seen by real-time threads.

6 Experimental Tests and Evaluation

The development process in industrial, high performance, and time-critical environments includes an extensive and thorough performance testing. It is important to build a test environment that resembles the operational environment as closely as possible, both in hardware and software, and to perform tests in adequately set up stress conditions.

6.1 Test Environment

The Operational Environment of an industrial time-critical application is mainly composed by "Command and Control" (C2) applications. We model a C2 distributed application as constituted by components of one of three types [9]: a

sensor type component that receives the data from the environment, an *elaborator* component that computes the actions to be taken in response to data received from sensor, and an *actuator* component that finally executes these actions in order to modify one or more entities of the environment to be controlled. Following this scheme, a distributed application was developed in order to realistically mimic this behavior. The application testing utilizes a fictional remote control of a vector in a 2D space along the two coordinates and speed. Such an application is computationally very similar to real-world C2 applications used in industrial, aerospace, and military contexts, because it involves geometric algorithms on 2D polygons as well as trigonometric and floating-point operations. Moreover, to perform stress tests, benchmark programs are used to synthetically generate the load that simulates extreme operational conditions. Stress tests are necessary for an industrial and mission-critical real-time system to check if the software retains its performance level even in presence of high or spike load. The *Ubench* and *Hackbench* benchmarks were used in order to study the behavior of the application under varying load circumstances.

6.2 Test Results

The test were performed with unloaded systems, with Ubench load (CPU and memory), and with different Hackbench load parameters. The goal was to determine how the performance of the distributed application as a whole was affected in scenarios of high load. Figure 1 shows the round trip time (RTT) of the distributed application in the Ubench-loaded configuration, *i.e.*, the time that occurs between the data capture from the *sensor* component and the *action* taken by the *actuator* component. The total RTT of the application roughly doubles under loaded conditions, due to the number of context switches needed between the benchmark program and the application, but never exceeds the value of 2.2 milliseconds, as reported in Table 1.

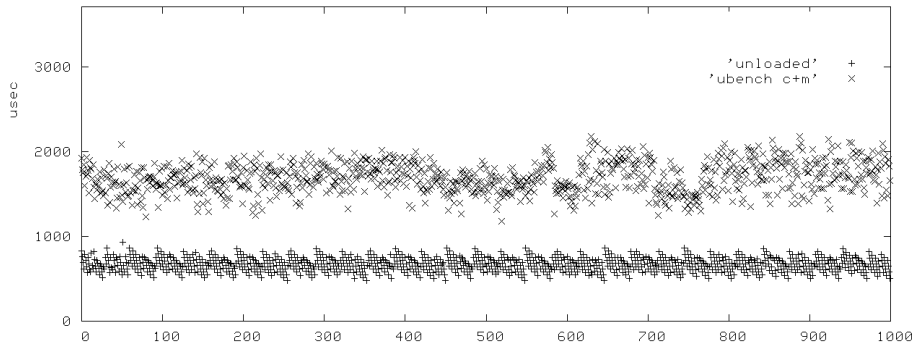

Fig. 1. Computational and Memory Load Sensitivity - Ubench

	Average	Standard Deviation	Maximum
Unloaded	669.67μs	81.62μs	932μs
Loaded	1694.39μs	178.45μs	2182μs

Table 1. Round Trip Time Latency Comparison - Ubench

In the Hackbench tests we measured mean, maximum, and standard deviation values of the application RTT while varying the coefficient of load (passed as parameter to the benchmark). The results (see Figure 2) do not show a significant relation between the rise of the load parameter and the RTT either in its maximum value or in mean and variance. It is worth noting that both load configurations were tested thoroughly with increasing loads until reaching instability of the host systems. Yet the performance of the distributed application was predictable and reliable, and retained real-time class performance.

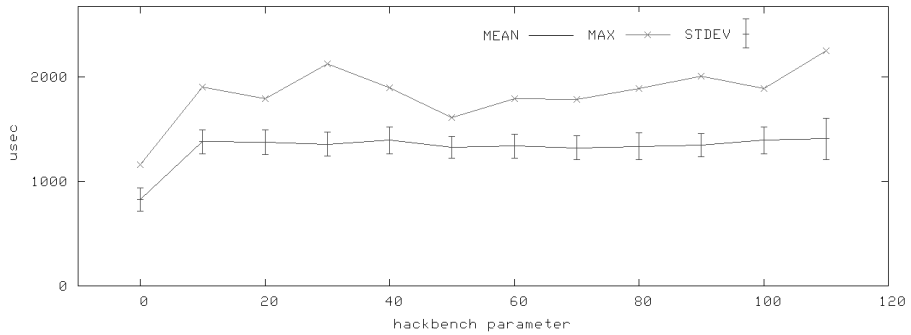

Fig. 2. Task Scheduling Load Sensitivity - Hackbench

7 Conclusions and Future Work

In this paper, we described recent research and development efforts in building an open source runtime and communication middleware layer for distributed real-time applications. Within this context, Harness represents an optimal choice of *base line* due to the extreme dynamic modularity its pluggable architecture offers. We exploited this feature to build the desired set of real-time functionalities in the form of *plugins* that realize communication and priority-aware execution services with a real-time level of performance.

The performed tests show how a distributed real-time application (in this case, a Command & Control application) can utilize the developed features. Our work, however, makes the assumption that the underlying software layers can provide an adequate level of performance. This is generally not true in the chosen test environment.

Future work in this area will include porting the Harness middleware layer onto a dedicated *hard real-time* operating system and network stack. Xenomai seems to be the best candidate of the set of real-time OS's, because it can provide access to (hard) real-time features, while keeping an external POSIX interface (Xenomai Skin Technology [10]). As an added bonus, Xenomai offers a complete real-time networking stack with its integrated RTNET technology. Another solution consists of adopting a completely different network technology that offers an entire stack of real-time-oriented features, like Infiniband and its related protocols. Ongoing research activities are conducted in this direction by the Applied Research & Technology Department of MBDA Italia S.p.a.

References

1. Bray, M.: Middleware, Software Technology Review at Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA. Available at http://www.sei.cmu.edu/str/descriptions/middleware.html (1997)
2. Geist, G.A., Beguelin, A., Dongarra, J.J., Jiang, W., Manchek, R., Sunderam, V.S.: PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, USA (1994)
3. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. MIT Press, Cambridge, MA, USA (1996)
4. Real-Time Innovations, Inc, Santa Clara, CA, USA: Data Distribution Service. Available at http://www.rti.com/products/data_distribution/ (2007)
5. Washington University, St. Louis, MO, USA: Adaptive Communication Environment (ACE) with TAO. Available at http://www.cs.wustl.edu/∼schmidt/TAO.html (2007)
6. Object Management Group, Inc, Needham, MA, USA: Data Distribution Service for Real-time Systems. Available at http://www.omg.org/technology/documents/formal/data_distribution.htm (2007)
7. Engelmann, C., Geist, G.A.: A lightweight kernel for the harness metacomputing framework. In: Proceedings of the 14^{th} Heterogeneous Computing Workshop (HCW) 2005, in conjunction with the 19^{th} International Parallel and Distributed Processing Symposium (IPDPS) 2005, Denver, CO, USA (2005)
8. Engelmann, C., Geist, G.A.: RMIX: A dynamic, heterogeneous, reconfigurable communication framework. In: Lecture Notes in Computer Science: Proceedings of the International Conference on Computational Science (ICCS) 2006, Part II. Volume 3992., Reading, UK (2006) 573–580
9. Ravindran, B.: Engineering dynamic real-time distributed systems: Architecture, system description language, and middleware. IEEE Transactions on Software Engineering **28** (2002) 30–57
10. Gerum, P.: Xenomai - Implementing a RTOS emulation framework on GNU/Linux. Available at http://download.gna.org/rtai/documentation/vesuvio/html/xenomai (2004)