DDRAFT: Supporting Dynamic Distributed Real-time Applications with Fault-Tolerance

Carlos Almeida
cra@digitais.ist.utl.pt
IST-UTL*

José Rufino
ruf@digitais.ist.utl.pt
IST-UTL

Paulo Veríssimo
pjv@di.fc.ul.pt
FC/UL†

1 Introduction

With the widespread use of computers and communication networks, there is a potential for the development of a new class of distributed applications that until now were not viable, or were only feasible in centralized environments. Some of these applications have requirements for fault-tolerance and real-time characteristics (e.g. distributed real-time databases associated with new telecommunication services). Although some new communication network technologies (e.g. ATM) have improved the synchronism properties of distributed environments, these are not always fully synchronous. They are at most *quasi-synchronous* [7]. Only a small part of the system can be considered as synchronous. The rest has a more dynamic behavior exhibiting, for a given activity, worst-case "execution"[1] times that are much higher than the normal "execution" times (see Figure 1 for the case of message delivery time). In this type of environment, the development of distributed fault-tolerant real-time applications is a difficult task. There is, however, a demand for it, and so, providing support for the development of such applications is of utmost importance. That is our goal with DDRAFT (Dynamic Distributed Real-time Applications with Fault-Tolerance).

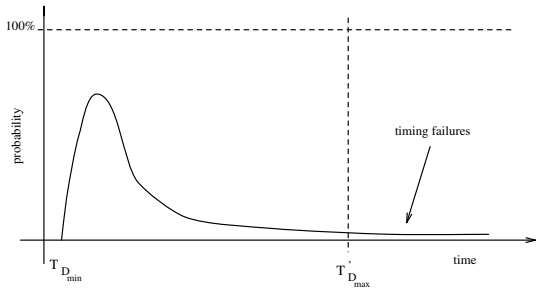

Figure 1: Distribution function for message delivery time (T_D) in a *quasi-synchronous* system. The worst-case is much higher than the normal case. Assuming as maximum a value closer to the normal case (due to practical reasons) increases the probability of having timing failures.

The main problem with the type of environment described above, is the difficulty to have efficiency and at the same time provide safety in a timely fashion. Having a system whose load is not completely controlled, or having applications which have dynamic characteristics that make it hard (if not impossible) to know *a priori* what are the worst-case scenarios, does not allow to have a resource adequacy policy, or at least such policy is not cost-effective for the target applications.

Not all real-time applications are implementable in such environment. If an application needs high assurance that all deadlines are met, because if not there will be high costs involved (loss of human lives or high cost components), then a resource adequacy policy must be used. However, some classes of real-time applications "can live" with occasional deadline misses, as far as those situations are handled in a timely and controlled way. They need safety in a timely

*Instituto Superior Técnico - Universidade Técnica de Lisboa, DEEC, Av. Rovisco Pais - 1096 Lisboa Codex - Portugal. Tel: +351-1-8418397 - Fax: +351-1-8417499.

†Faculdade de Ciências da Universidade de Lisboa, Portugal.

[1] Transmission in the case of a network related activity.

fashion. That is what we aim to provide with the DDRAFT system.

The paper is organized as follows: in the next section we briefly address common approaches to real-time, namely hard and soft real-time. In Section 3 we present our *quasi-synchronous* approach. In Section 4 we present the main goals of DDRAFT. The paper ends with some notes about the current status and future work.

2 Hard and soft real-time

In the literature related to real-time, one can find mainly to types of real-time systems (or two approaches) – hard real-time and soft real-time. Although there is not a perfect consensus about their definitions, a hard real-time system is said to be one where missing a deadline means a very high cost (loss of human lives or high cost components), whereas in a soft real-time system it is acceptable to miss some deadlines.

The way these two types of system are built reflect the goals they want to achieve. In the case of a hard real-time system it is necessary to provide strong guarantees about timeliness. The time domain must be addressed in such a way that all deadlines are met. This implies the use of a resource adequacy policy and the system must be designed considering the worst-case scenario. Load must be fully controlled and one must have available all the resources needed for a worst-case situation. This implies a high cost and usually restricts this type of approach to small scale systems where it is feasible to have a controlled environment. System behavior must be completely pre-defined in a static way at design time so as to make sure that the correctness in the time domain is achieved. This is the type of system that is usually used in safety critical applications such as the control of a nuclear power plant, for example, where there is a need for both safety and timeliness, and deadlines must not be missed.

On the other hand, in the case of soft real-time systems, the approach is more probabilistic. There is usually a great concern about throughput trying to optimize the mean values, and having only probabilistic guarantees that no more than a specified percentage of deadlines will be missed. The timeliness is not completely controlled. Once in a while there can be timing failures, which is supposed to be acceptable by this type of applications. These applications can be more dynamic, and they try to achieve a cost-effective way of working. Timeliness is not fully granted and safety may not be fully granted. In order to achieve safety it is sometimes necessary to introduce long delays. This approach can not be used by safety-critical applications, but it is acceptable for some other applications such as applications in the area of multimedia that manipulate video and audio, for example.

3 The quasi-synchronous approach

New applications requirements

With the widespread use of computers and new communication network technologies referred above, there is however a demand for new real-time applications with requirements that are not well addressed by the two approaches previously presented. They cannot afford the cost of a hard real-time approach (or due to the dynamic characteristics of the application and environment it is not possible to use such approach), but they need more guarantees than those provided by a common soft real-time approach.

Applications such as the ones associated with new telecommunication services that need to use real-time databases, may have requirements that are not fully addressed by a traditional soft real-time approach (even if for some services this approach is enough). When there is a need for high availability and the application is dynamic and has both timeliness and safety requirements, one must have some form of validation. Although a hard real-time approach is not cost-effective, a pure soft real-time one is not enough. It is acceptable to have some timing failures, but that situation must be handled in a controlled and timely fashion. Applications such as distribut-

ed real-time databases that use active replication and where dynamic updates are done, are examples of applications with such requirements. This type of application may be found in the area of new telecommunications services, for example.

In order to solve the problem presented above, it is necessary to address both the aspects related to communications and operating system. It is necessary to obtain an architecture and the mechanisms able to provide the desired guarantees in a *quasi-synchronous* system.

The quasi-synchronous model

In a *quasi-synchronous* system, the system can be modeled as if it was a synchronous system, in the sense that there are bounds on process speed, message transmission delay and local clock rate drift, but some or all of those bounds are not precisely known, or have values that are too far from the normal case, that in practice one must use other values (closer to the normal case). In both cases it means that there is a non-null probability that the values we pick are not correct. This is a realistic scenario when there are situations of overload. Better synchronism properties are restricted to a small part of the system: a few high priority activities, and a small bandwidth channel for high priority messages.

By restricting synchronism properties to specific system modules we are able to obtain the desired properties in a larger setting. These modules are used to validate and control the other parts of the system thus making it possible to achieve safety in a timely fashion. This approach is specially useful to mission-critical applications.

4 DDRAFT main goals

With DDRAFT we aim to provide system support for the development of fault-tolerant distributed real-time applications that run on environments that are not completely synchronous. This implies the design of a modular architecture that tackles both the aspects related to processing (CPU) and communications (Network). Using the *quasi-synchronous* approach we plan to build a system where, although there is not a complete control of load, it is possible to have some restricted components with "good" synchronism properties. These components are used to validate the rest of the system in order to achieve safety in a timely fashion.

DDRAFT offers a set of group communication protocols suitable for real-time in *quasi-synchronous* environments. The user can choose among several different qualities of service (QoS). These QoS are based on basic properties, namely, agreement, order and timeliness.

Group communication protocols and group management are a powerful tool to support the development of applications that require the use of replication. By using a hierarchical structure for those groups, it is possible to have a group layer that allows an efficient handling of timing failures. DDRAFT uses two layers for group management. At low level there is a basic group layer (BG) corresponding to traditional group management (a participant leaves the group when it wants, or when it crashes). On top of the basic group layer there is a layer with light weight groups (LWG). A participant may leave the light weight group even when the basic group is still operational. This makes it possible to efficiently enforce safety properties when timing failures are detected. Keeping the basic group operational reduces recovery time.

DDRAFT uses a timing failure detection service (TFDS) to disseminate with timeliness guarantees control information that is used by the communication protocols to validate their properties [2]. The implementation of TFDS may require the use of a specific network or a dedicated channel in a more generic network. We are exploring several approaches. Depending on what is available and on the existing resources for a given application one may choose the approach that best fit in a given scenario. If an ATM network is available, for example, TFDS can be implemented using a channel with better guarantees than the channels used for normal communication. In the situation where there is only available a network

such as Ethernet, then, if the temporal granularity of the application is large enough, it is still possible to add some form of access control at an upper level and being able to obtain the desired properties. Another approach concerns the use of a specific network. We are experimenting with the use of a Controller Area Network (CAN) [5] (this approach is specially interesting for applications in the area of automation control, for example). This network can be used together with a more generic network such as Ethernet. Ethernet can be used for normal messages, and CAN is used to implement TFDS and other services requiring better synchronism properties such as clock synchronization, for example. A somehow similar configuration has been used by other research groups [4]. They use a CAN network in conjunction with a fiber optics network. CAN is used for arbitration and data is sent via fiber optics.

Besides the aspects related to communications, it is also necessary to address the aspects related to processing. In the new type of environment described above, the considerations made about communications also apply to processing. In this new dynamic environment there is a demand to try to put together applications that are real-time and applications non-real-time. Moreover, people want to continue to use generic operating systems (that they are used to, and are suitable for generic applications that they still need), and at same time have applications that have real-time requirements [6]. Although some soft real-time applications can fit in this scenario, not all of the desired applications are viable without adding some real-time extensions to the generic operating system (OS) used.

In DDRAFT we want to explore several approaches to these real-time extensions. From a simple high priority component integrated in the OS to the use of a co-processor with or without a specific real-time OS. The main idea being similar to what is done by TFDS in what concerns communications: to have a component able to make timely validations of what is done by the generic OS.

5 Current status and future work

In DDRAFT we started by addressing the aspects related to communications. TFDS and some group communication protocols are already implemented [1, 2]. As the complete system is not available yet, those protocols were validated using simulation. The results obtained so far are encouraging [3].

We are now addressing the aspects related to processing in order to obtain a prototype. This will allow a more realistic type of utilization. We plan to develop example applications to show the potential of the DDRAFT system as a support for the development of fault-tolerant distributed real-time applications that run on dynamic environments.

References

[1] Carlos Almeida and Paulo Veríssimo. An adaptive real-time group communication protocol. In *Proceedings of the First IEEE Workshop on Factory Communication Systems*, Leysin, Switzerland, October 1995.

[2] Carlos Almeida and Paulo Veríssimo. Timing failure detection and real-time group communication in *quasi-synchronous* systems. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems*, L' Aquila, Italy, June 1996.

[3] Carlos Almeida and Paulo Veríssimo. Timing failure detection service: Architecture and simulation results. Technical Report CTI RT-97-05, Instituto Superior Técnico, Lisboa, Portugal, December 1997.

[4] A. Burns, N. Audsley, and A. Wellings. Real time distributed computing. In *Proc. of the 5th Workshop on Future Trends of Distributed Computing Systems*, pages 34–40, Cheju Island, Korea, August 1995. IEEE.

[5] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. Technical Report CTI RT-97-04, Instituto Superior Técnico, Lisboa, Portugal, December 1997.

[6] J. Stankovic. Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, 28(4):751–763, December 1996.

[7] Paulo Veríssimo and Carlos Almeida. Quasi-synchronism: a step away from the traditional fault-tolerant real-time system models. *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS), IEEE Computer Society*, 7(4):35–39, Winter 1995.