Applying Feedback Control to QoS Management

Giovanna Ferrari
Distributed Systems Group, Department of Computing Science,
University of Newcastle upon Tyne,
United Kingdom

Giovanna.Ferrari@ncl.ac.uk

Introduction

The importance of Feedback Control Theory used for dynamic systems has grown in recent years, due to the fundamental role played in modern technological systems. It represents a well developed analytic foundation for performance control in physical systems. Recent results have shown that the Feedback Control can also be successfully applied to the control of software performances of open systems executed in distributed environments.
Many modern applications require some form of performance assurances such as guarantees on timeliness, availability, bandwidth, data consistency, or jitter, all of them non-functional requirements of Quality of Service (*QoS*).
We believe Feedback Control Theory can be the theoretical basis for the design of adaptation-based middleware architectures that handle QoS-aware services for modern real-time systems, which include online trading, e-business servers, auctions and application service providers.
In this paper, introducing classical Control Theory used for dynamic systems' control, we explain the approach of the Feedback Control recently applied in computing science. We illustrate some related works and a model of middleware for QoS-control, that represents a meaningful example in the direction of the design of an adaptation-based architecture.

Dynamic systems and classical Control Theory

The Internet presents a convenient interface for performing online trading, auctions and other e-commerce related activities, all of which are emerging performance critical applications. In such application, failure to meet performance specifications may result in loss of customers, financial damage or liability violations.
These applications are essentially soft real-time applications and each of them can be seen as a dynamic system living in an ever changing environment.
In its most general meaning, the term *system* refers to some physical entity on which some action is performed by means of an input, the system reacts to this input producing an output. A *dynamic system* is a system whose phenomena occur over time. One often says that a system *evolves over time* [1]. An instance in Physics is the simple pendulum; likewise an e-business server can be in the Computing Science field; in it the resource requirements and the arrival rate of service requests occur over time, however it is even more difficult to model because none of them is known *a priori*.
Problems in complex and dynamic systems include the difficulty of modelling them precisely and potential lack of convergence to desirable stable states. There are many reasons why a dynamic system may not be able to stabilize on an expected state by itself, either there is disturbance from the environment, or the limit of its structure makes it impossible to converge automatically. These problems call for mechanisms that can control them effectively, without depending on detailed insight into their internal structure or on precise models of their behaviour.
Physical and engineering sciences have a well developed analytic foundation for performance control in physical systems and it is based on *Control Theory*. It employs differential equations as a fundamental modelling tool for the design of a control system, where, generally speaking, the objective is to make some output behave in a desired way by manipulating some input [2].
The subject of the study is the *control system,* consisting of a controlled system in combination with a controller. The interactions between the controlled system and the controller consist of observations and manipulations performed by the controller on the controlled system. In *Fig.1* is shown the building blocks of the control process.

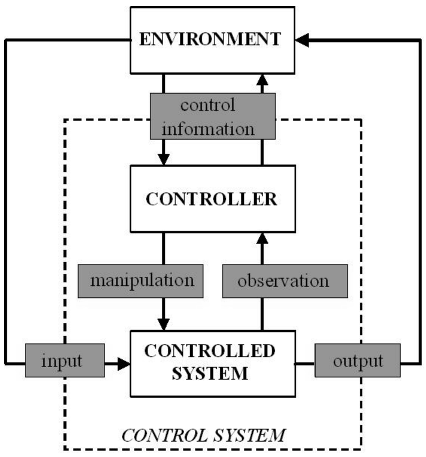

Fig. 1: Building blocks of the control process

Control systems exist in a virtually infinite variety both in type of application and level of sophistication. For example, the water heater in a house is a system in which the sign of the difference between desired and actual temperatures is used for control. Through the *observations*, the controller knows if the temperature drops below a set value, given by the *control information*, say a thermostat, so it switches on the constant heat source of the controlled system, executing a *manipulation*; it switches it off again when the temperature rises above a set maximum,; the *input* and the *output* of the system are respectively the cold and the hot water [3].
The relationship between the controlled system and the controller can be realised using different approaches. The traditional approach in the field of computing systems has been to quantify hardware speed and software execution requirements, then apply an appropriate combination of pre-run time analysis, admission control, and resource allocation algorithms to ensure that the system is not overloaded and that the desired performance is achieved [4]. This is the *Feed-forward Control* strategy, where manipulation through control actions is determined based on observation of the input to the controlled system. The approach relies on *a priori* workload and resource knowledge and it works well for embedded computing systems, such as avionics and factory automation, in which an accurate model exists for the system *a priori*. Whereas the *Feedback Control* strategy can be applied for behaviour optimisation in unpredictable or poorly modelled environments; according to this strategy, the measurement of the output delivered by the controlled system is compared with a reference and the difference between them is used by the controller to decide on the control action to be taken, regulated according a continuous performance feedback.

Feedback Control

A Feedback Control system is based on a set of control related variables and a *feedback control loop* that continuously monitors the behaviour of the controlled system, as shown in *Fig.2*.
The loop compares the actual behaviour against a specification of the expected behaviour, adjusting it accordingly to ensure respect of the behavioural performances. The main components are:

1) *Monitor*: the component that measures the controlled variable and feeds the sample back to the Controller.
2) *Controller:* the component that compares the performance reference with corresponding controlled variable to get the current error. If the error is rises above a set maximum, the Controller calls an algorithm to compute the control input, the new value of the manipulated variable based on the error.
3) *Actuator*: the component that changes the manipulated variable based on the newly computed control input, given by the Controller.

For instance, using the Feedback Control for scheduling purposes, the Actuator may implement a mechanism that dynamically reallocates the resource corresponding to the manipulated variable; say that the total requested CPU utilization is the variable, the Actuator dynamically adjusts tasks according to a scheduling policy.
There are some key variables that represent the main performance metrics' values of the whole system:

1) *Controlled variable*: the performance metric that characterizes the system performance defined over a sampling period and it is controlled by the system in order to achieve the

desired performance. The choice of controlled variables depends on the performance guarantees that need to be provided to the specific application of a system. For example, if an absolute delay guarantee is required in an Internet server, as critical stock trading operations in an on-line trading system, the absolute service delays of HTTP requests should be defined as the controlled variable. Some typical controlled variables could be the deadline-miss ratio and the CPU utilization, measured there where explicit timing constraints need to be respected, for soft real-time systems, like multimedia streaming, process control, and robotics.

2) *Reference*: the variable that represents the desired system performance in term of a controlled variable. The performance reference defines a contract established between the adaptive resource and the users such that the performance reference should be enforced. The difference between the performance reference and the value of the corresponding controlled variable is called the *error*.
3) *Manipulated variable*: the system attribute that is dynamically changed by an Actuator. The manipulated variable should be effective for performance control, in other words, changing its value should affect the system's controlled variables. The choice of manipulated variable should reflect the resource bottleneck of a system. For example, the total requested utilization should be used as a manipulated variable if CPU is the bottleneck resource of a web server [5].

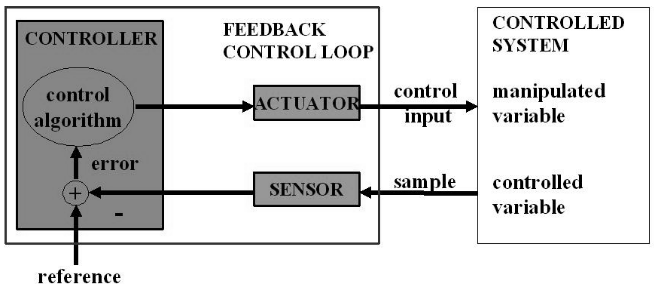

Fig.2: Feedback Control System

Since it is impossible to achieve absolute guarantees in a system where load and resources are not known *a priori*, Zhang et al. [6] define the absolute guarantee problem as one of *convergence* to a specified performance. The statement of the problem is to ensure that a performance metric converges within a specified exponentially decaying envelope to a fixed value and that the maximum deviation be bounded at all times.

Feedback Control Theory can be used as analytic engine to provide robust convergence assurances in every dynamic system, as well as performance-critical software applications, such as mail servers, web servers, and proxy caches. In order to utilize it in these applications, three main design challenges have to be solved. First, from a Control Theory perspective, a general methodology needs to be developed for converting QoS specifications of a computing system such as an web server, into feedback loops with known set points and feedback control parameters. Second, from a systems perspective, a convenient interface needs to be found between the service software and the middleware control loops that manage its performance. Third, appropriate software performance sensors and actuators must be designed.

Recently, there has been a lot of resurgent interest in Feedback Control Theory and it has been successfully applied to a several computer system projects. At the network layer, Hollot et al. [7] applied Control Theory to analyze the RED congestion control algorithm on IP routers. In the area of CPU scheduling, Steere et al. [8] developed a feedback based CPU scheduler that synchronizes the progress of consumers and supplier processes of buffers. Recently web server software has become a focus area of Feedback Control because the unpredictability of the workload. Examples of such QoS control include delay and bandwidth control in web servers [9] and queue management in e-mail servers [10], this one used at IBM laboratories to implement a controller for Lotus Notes system.

Middleware for QoS Control

In our researches we are investigating the field of QoS-aware middleware, in particular for the purposes of the area of Application Services Provider (*ASP*).

Actually organisations are increasingly focusing on their core businesses and streamlining their operations by ‘outsourcing’ non-core businesses to external organisations, and many organisations find it cost effective to outsource their applications to ASP. An ASP typically uses middleware and component technologies for deploying, hosting and managing applications and it must be able to meet QoS agreements of hosted applications. Such agreements are established with customers in terms of Service Level Agreement (*SLA*) that defines the responsibilities of an ASP and its users. The SLA also identifies the service provided as well as the supported products, measurement criteria, reporting criteria and quality standard of the service [13].

We believe that the application of Feedback Control Theory can lead the design of the middleware architecture capable to control and ensure the performance metrics given by the SLAs’ specifications.

There are some interesting related works where the Feedback Control Theory has been successfully applied on the middleware architecture for the QoS control.

One of these is ControlWare [6], middleware architecture for QoS guarantees in distributed environments such as the Internet, planned at the University of Virginia. The architecture implements a new paradigm for QoS control, suitable for systems operating in highly uncertain environments or when accurate system load and resource models are not available. It is a sort of comprehensive middleware service that incorporates the principles of QoS control with the Feedback Control Theory under a well defined set of APIs, which substantially reduces the development effort of performance assured applications and Internet services.

The SWiFT project [11] shares some similar goals with ControlWare. It is a toolkit for constructing feedback control loops from libraries that allows easy dynamic reconfiguration of components by limiting the interaction between components to a simple input/output model and by supporting guarding and re-plugging of controllers. This reconfiguration allows the application to adapt efficiently across a wide range of operating conditions.

Another significant example of model of architecture, which key part is the QoS-control system, is designed within the AMIDST project, developed by the University of Twente[12]. The QoS controller in this system observes and, if necessary, manipulates the state of the controlled system, represented by the middleware platform that supports distributed applications. The design of the QoS controller is an architectural framework based on models from Feedback Control Theory. This ensures its stability with respect to evolving requirements, and its applicability to a wide range of controlling techniques. The middleware platform encapsulates the computing and communication resources at each individual processing node, which may be manipulated in order to maintain the agreed QoS.

The picture in *Fig. 3* shows the specialisation definition of the generic control model for controlling the QoS provided by a middleware. Here two symmetrical structures can be identified, one for handling QoS measurement concerns and another for handling QoS manipulation concerns. The *probes* are points of observation or manipulation available or planted in the middleware platform. The *sensor* is the mechanism that uses a probe to obtain *observations*, which are interpreted in terms of measurements in order to be compared with the reference. For instance, observations can be time moments of the sending of a request and the receiving of the corresponding response. The needed measurement could be the average response time, which implies that the average of the difference between the time moments observed should be calculated in order to generate the measurement and the calculation is performed by an *interpreter*.

A *comparator* compares the measurement and an associated reference value, the *agreed QoS measure*, determining the *difference*. A *decider* gets the difference and applies some algorithm to establish a *control strategy*, consisting of the objectives to be reached in this execution of the control loop.

A *translator* is responsible for translating the control strategy to a collection of *control actions*, i.e., manipulations of the controlled system and an *actuator* schedules them such that they are carried out using one or more probes.

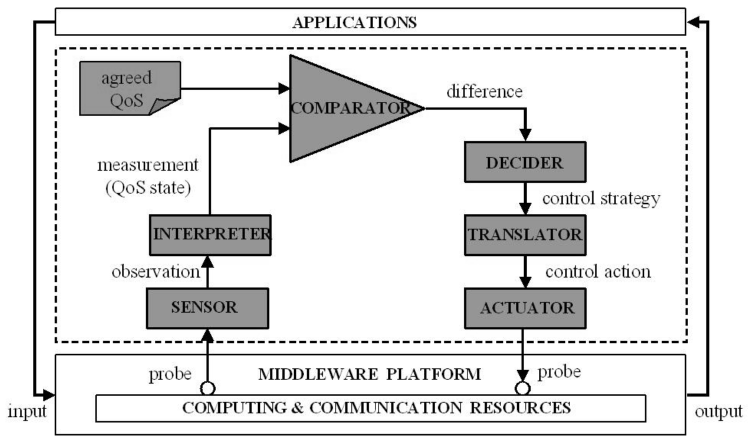

Fig. 3: Diagram of a model of middleware for QoS control

This is a meaningful example in the direction of the design of adaptation based middleware architecture; it observes the behaviour of the system and reconfigures itself to provide the required behaviour.
In general, adaptation-based approaches involve implementing mechanisms that periodically monitor the resource availability of the execution environment, and provide the applications with appropriate adaptation and reconfiguration properties. Adaptation-based middleware architecture demands the following three principal stages of run time support: *probing* the performance of QoS parameters, *instantiating* the initial middleware configuration, and *adapting* to on-the-fly variations.
The analysis shown can guide the development of Trusted and QoS-aware middleware services for Application Hosting, employing Feedback Control Theory to govern component execution.
This is the part of the research done in the domain of the TAPAS project, that involves a consortium of five sites across Europe. The TAPAS's overall objective is to develop novel methods, tools, algorithms and protocols that support the construction and provisioning of Internet application services capable of meeting specific non-functional requirements like fault tolerance, availability, security and timeliness. The project will achieve the overall objective, among other things, by developing QoS-enabled middleware services capable of meeting SLA between application services and will enhance component based middleware technologies such that components can be deployed and interact across organisational boundaries.
In the design of the middleware architecture for this research there are three main basic QoS services, fundamental in order to provide end-to-end QoS: the *Configuration Service*, applied for the QoS negotiation, the resource discovery and reservation; the *Trust Manager Service*, supervising the management of the contracts and the trust relationships; the *Run Time Support Service*, in charge of the resource release and the QoS monitoring and adaptation [14], and its theoretical basis can be constituted by the Feedback Control Theory.

Conclusions

Within this paper we have expounded an introduction to the application of Feedback Control Theory to QoS management.
We have shown how Feedback Control can offer a valid solution to the problem of achieving performance guarantees for modern soft real-time applications requiring performance assurances as guarantees on timeliness, bandwidth, data consistency, or jitter. It has been applied by the engineering community in controlling a vast variety of physical systems and its success is due to the robustness in the face of modelling errors and external disturbances, which reduces the need for accurate system models, a much welcome property when accurate models are difficult to construct.
Here are presented some examples of recent results that prove that Control Theory can also be efficiently applied to the control of software performance in unpredictable environments like Internet. In particular we believe it can be the theoretical basis for the design of a QoS-aware middleware architecture that addresses one of the targets of the TAPAS project.

References

1. Mathematical Methods for Robotics and Vision, *C. Tomasi, Stanford University.*
2. Feedback Control Theory, *J. Doyle, B.Francis, A. Tannenbaum, Macmillan Publishing Co.*

3. Feedback control systems, *J.van De Vegte, Prentice Hall International Editions.*
4. Hard Real-Time Computing Systems, *G. C. Buttazzo and J. A. Stankovic, Kluwer Academic Publishers.*
5. Feedback Control Real-Time Scheduling, *C. Lu, PhD Thesis at University of Virginia, May 2001.*
6. ControlWare: a Middleware Architecture for Feedback Control of Software performance, *R. Zhang, C. Lu, Tarek F. Abdelzaher, and J. A. Stankovic, International Conference on Distributed Computing Systems , July 2002.*
7. A Control Theoretic Analysisy of RED, *C.V. Hollot, V. Misra, D. Towsley and W. Gong, IEEE INFOCOMM , 2001.*
8. A feedback-driven proportion allocator for real-rate scheduling, *D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu and J. Walpole. In Operating Systems Design and Implementation, 1999.*
9. Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach**,** *T. Abdelzaher, K. G. Shin, N. Bhatti, IEEE Transactions on Parallel and Distributed Systems, 2001.*
10. Using control Theory to achieve service level objectives in performance management, *S. Parekh, N. Gandhi, J. L. Hellerstein, D. Tilbury, T. S. Jayram, and J. Bigus, IFIP/IEEE International Symposium on Integrated Network Management, 2001.*
11. Swift: A feedback control and dynamic reconfiguration toolkit, *A. Goel, D. Steere, C. Pu, and J.Walpole, Technical Report at 2nd Usenix Windows NT Symposium, 1998.*
12. A Qos control architecture for object middleware, *L. Bergmans, A. van Halteren, L. Ferreira Pires, M. van Sinderen, M. Aksit, 7th Intl. Conf. on Interactive Distributed Multimedia Systems and Telecommunication Services, 2000.*
13. Application Hosting Requirements, *W. Beckman, M. Oleneva, Dortmund, May 2002.*
14. TAPAS Architecture: QoS-aware Containers, *V. Ghini, G. Lodi, N. Mezzetti, F. Panzieri, Newcastle upon Tyne, September 2002.*