An Adaptive Thin-Client Robot Control Architecture

Tim Edmonds[1] Steve Hodges[2] Andy Hopper[1,2]

[1] *Laboratory for Communications Engineering*
Department of Engineering
University of Cambridge
Trumpington Street
Cambridge, UK
tme23@eng.cam.ac.uk

[2] *AT&T Laboratories Cambridge*
24a Trumpington St
Cambridge, UK
{seh,ah}@uk.research.att.com

Abstract

This paper describes an architecture and runtime system to implement distributed control and data processing applications in a thin-client manner, suitable for implementing a thin-client mobile robotics system. The system varies control fidelity and locality to adapt a control application to changes in Quality of Service availability and processing resources using a cost benefit model.

An example application is presented in which the architecture is used to implement the distributed control of an inverted pendulum over a shared network. Performance results are compared with non-adaptive distributed control approaches.

1. Introduction

There are many situations in which several mobile robots must work together to achieve a united goal. The control of such systems is typically performed locally in a distributed manner to ensure a simple, scalable, yet robust system. However, such distributed control systems are less efficient and more restricted than a centrally served system. In many situations, a control system with a global view is required where a local view can not provide sufficient information for operation[7].

A thin-client control architecture is useful for multi-robot systems as the robots can be made to be cheap, plentiful and lightweight (in all aspects: mass, interface and power consumption) while a powerful server can control several such robots centrally. However, such a system is vulnerable to the vagaries of wireless and network communications as it requires bandwidth and latency *Quality of Service* (QoS) provisions.

While thin-client systems offer high maintainability and portability and low cost, the price of such a configuration is a strong reliance on communications infrastructure and characteristics. For control applications, the raw signals to and from the sensors and actuators will often require greater bandwidth than higher level control abstractions. Additionally, communication latency becomes significant and often unpredictable. While some networks can provide some QoS guarantees, these are expensive in relation to a shared network. For mobile systems that use radio networking, the communications characteristics can vary widely and rapidly.

2. Adapting a Control Application

To implement a real-time control system in a thin-client manner, we can take advantage of several distinguishing features of such a system to alleviate these network restrictions. Similar work in adaptation of distributed multimedia applications[1, 2]largely comprises varying the quality of the data by adjusting bit rate and resolution. Control and multimedia applications are similar in that they are both *analoguesque* in nature – the data involved is rarely discrete in nature and can tolerate degradation of rate or fidelity. However, while multimedia applications tend to focus on end-to-end streaming and experience a latency bounded by the performance of the network, control applications characteristically operate in a loop where the sensors and actuators are physically colocated but logically seperated by the processing. Hence, a control application can theoretically achieve zero latency by shortcutting the network. This suggests another method to combat latency in the application: move processing to the client.

However, the thin-client mantra says that processing should be performed on the server – thus, only lightweight processing could be performed on the client. Experience with thin client systems such as for robot football[5] has

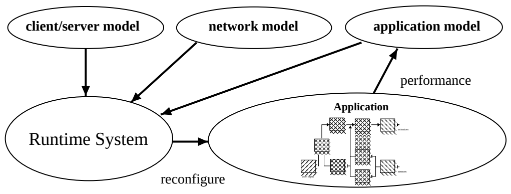

Figure 1. Runtime support in place

shown that even after sensor, actuator and communications processing is completeted, often spare processing cycles are available for lightweight application use, so low quality processing could be performed locally during some periods.

Hence, the above points present two variables for adapting a control application to best operate given the resources available to it: quality and locality.

2.1. Building an Adaptive Application

An adaptive application requires fine grained control of application quality and of application locality so that its operation may be transferred between client and server in a fine, but discrete way. Additionally, the application must have a runtime mechanism to balance the use of available resources by reconfiguring the application.

The application is specified in an abstract manner to the runtime as a graph of *functional elements* or *modules* which can then be manipulated. Design techniques such as dataflow analysis can be used to describe the application as a possibly branching chain of functional elements and is consistent with popular building block approaches. Similarly, Brooks' subsumption architecture[4] is an existing modular robot control design method that may be suitable for implementation in this manner. Isolating the implementations of the functional elements allows the runtime to hot-swap or migrate modules dynamically.

The application runtime, shown in Figure 1, must dynamically configure the application for maximum performance at least cost. It uses models of the application, the network performance, and the client and server machines to determine the best configuration of functional elements in terms of exact implementation and locality. It uses feedback from the application proper to determine the current performance level. A cost-benefit optimisation is used to calculate the dynamic application reconfiguration requirements. Every resource is given a cost function (in dollars) which may vary according to the demand and availability of the resource. The application modules provide models which indicate the benefits attainable from any given configuration. In this way, the runtime can optimise the configuration for best performance given a particular application budget or calculate a least cost configuration to attain a given performance level. Application feedback adjusts the module models to inflate or deflate costs to reflect current performance levels. Using these costs and benefits, the runtime cuts the application graph of functional elements to dynamically divide the application between client and server in the most cost-effective manner.

Specifying resource costs in currency terms is an important feature for deployment of systems which might spread over large geographical areas. In these situations, much of the infrastructure may not be owned by the system operator but leased from other agents. This may apply to both processing and network resources. Already we are seeing situations where processing can be bought from third party operators and network capacity of different QoS characteristics may be bought or sold dynamically with pricing based on demand. Today there exists bandwidth exchanges such as Band-X and Arbinet which trade in telecommunications capacity on a minute basis and it is easy to see a future where bandwidth everywhere is bought and sold on a very fine time scale. An application could then buy bandwidth of strict QoS characteristics when performance requires it and but then use poorer QoS internet communications when its requirements are less.

Less tangible resources such as battery power are harder to value in hard currency but can be approximated, perhaps with user input. Similarly, the performance benefit gained from a particular configuration is difficult to quantify and is inherently application dependent and so must be provided by the application itself. The runtime can monitor network and processor usage but must rely on the application to provide performance clues.

3. Controlling an Inverted Pendulum

Control of an inverted pendulum system is a classic control problem with hard real-time characteristics. The control aim for this system is to maintain the pendulum in an inverted, upright position by applying forces to the trolley. While controlling such systems remotely or collectively has no clear advantage, its hard real time nature produces clear quantifiable results. The system is highly unstable and is very sensitive to small control loop timing fluctuations. This high susceptibility means that control over distance or over a generic network has proved to be ineffective without real-time network infrastructure and delay compensating control algorithms[8, 6].

To distribute a pendulum controller in a thin client manner, the control algorithm is implemented as a single fully migratable processing module and a pendulum interface module which is of course constrained to operation on the client only. Other ancillary modules provide user control and feedback. This configuration is shown in Figure 2. The migratory module hence has different implementations for client-side and server-side operation to reflect the low

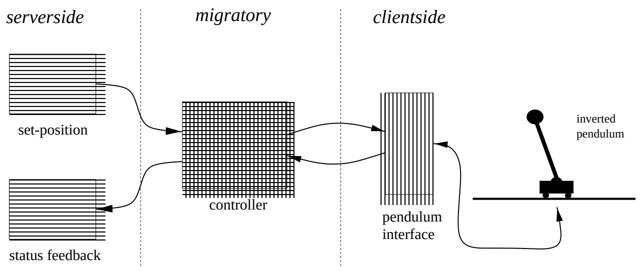

Figure 2. Controller modules

processing and low latency characteristics of the client. As the application processing is broken into only one stage, the granularity of adaptation is coarse but still demonstrates the utility of adaptation.

For the client side, a simple proportional feedback control loop is used in which the force feedback is a value directly proportional to the angular deviation of the pendulum from upright. This algorithm requires only three arithmetic operations per cycle which is easily achievable even on thin client hardware. The algorithm is able to take advantage of the known and small control latency to centre the pendulum with only small oscillations but is unable to handle more difficult conditions such as might occur when the trolley reaches the limits of its track and which may cause the pendulum to tip into an unrecoverable failure mode.

Timing measurements revealed that the one way latency between client and server varied between 0 and 8 sample periods, or up to 16 sample periods for the round trip. The server's control algorithm must be robust enough to be able to compensate for these discrepancies. For these reasons, the server implements a variation of Sutton et al.'s two-stage neural-net-like ACE/ASE reinforcement learning controller[3]. This algorithm is both processor and memory intensive in a manner that would make it unsuitable for implementation on a thin client device – analysis reveals that it requires approximately 2048 operations per sample and a storage area of approximately 1024 floating point variables.

The proportional controller gives reasonable performance at low latency while the learning controller gives good performance even with jittering latency. Hence, a suitable adaptation policy would favour the client-side controller when communications performance is poor and server-side if the pendulum is near the ends of the tracks or the pendulum is reasonably stable but not held exactly in place. A hysteresis function prevents rapid oscillations between client and server control at boundary conditions.

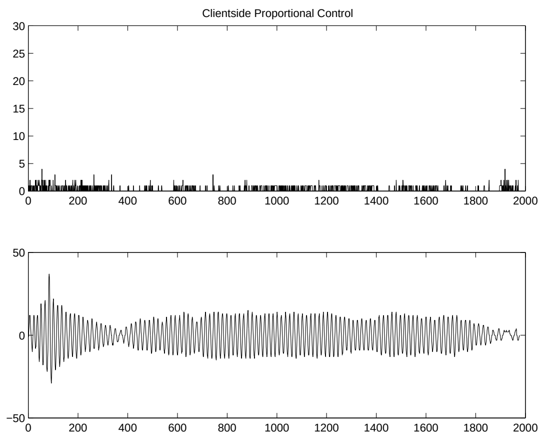

Figure 3. Client-side only operation showing latency (top) and error

3.1. System Performance

The application was deployed on as three distinct processes communicating via CORBA: simulated pendulum, application client and application server. Tests were performed for the application running in client mode, in server mode and in dynamic adaptive mode.

Figure 3 shows the system in pure client mode where the control algorithm module is the simple proportional controller located on the client. Here, latency is low (average 0-1ms) and jitter is low so the simple controller performs consistently with a steady-state rms error of 7.8 degrees from upright.

Figure 4 shows the system in pure server mode where the control algorithm module is the learning controller located on the server. The learning characteristic of the algorithm is visible as the high error region near the start point. In this case, the latency varies rapidly between 1ms and 21ms. Operating the simple proportional controller under such timing jitter conditions is infeasible and produced no recordable results. With the learning controller, after the learning process has stabilised, the rms error is approximately 6.2 degrees.

Figure 5 shows the system using application performance feedback to dynamically reconfigure itself. Included is a plot of client-side activation where a 1 indicates execution on the client and a 0 indicates server-side execution. As the ACE/ASE controller performs poorly at first while learning, the adaptation mechanism shifts control towards the client, adversely affecting the learning process. However, after this slow start, the system settles into primarily server-side operation. Sudden spikes in the error measurement result in transfers client-side for rapid correction after which control reverts server-side. Using this adaptation, the

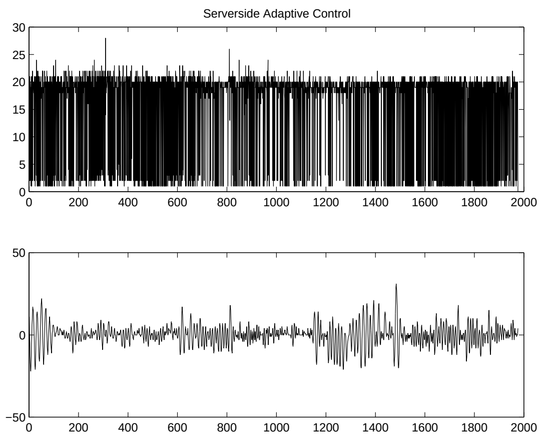

Figure 4. Server-side only operation showing latency (top) and error

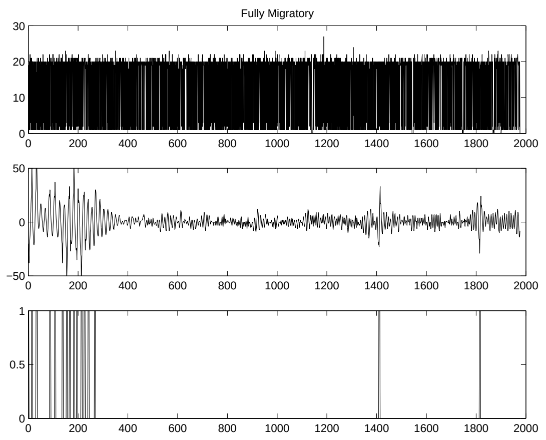

Figure 5. Adaptive controller operation showing latency (top), error (middle) and client-side activation (lower)

rms error (after initial learning) is 4.8 degrees, an improvement of almost 24% over the purely server based controller.

4. Conclusion

This paper has described a framework that provides a mechanism for adapting an application to the available processing and network resources by dynamically reconfiguring the locality and fidelity of operation. The framework itself does not provide quality of service guarantees but oversees the use of any guarantees that do exist such that optimal performance can be attained for minimal cost.

The example application of the distributed control of an inverted pendulum, while highly contrived, illustrates that dynamic reconfiguration of application fidelity and locality can be used to effectively combat latency and other resource issues in a distributed real-time system.

Further work is currently underway to extend this framework to provide generic cost-benefit based reconfiguration to mobile applications. This takes the concept of adapting to QoS provisions to also adapt to other varying resource constraints such as processing capability, network availability and power consumption. This is particularly aimed at mobile or wearable devices which by their nature must be lightweight with limitted power and may have varying levels of network access as they move between access mediums. An example of such an application, currently under development, is a speech recognition user interface. Operating on such a mobile device, the application leverages the superior resources of a distant server to perform high quality speech to text translation while gradually reverting to client processing as network connectivity becomes more expensive or the server becomes more loaded.

References

[1] O. Angin, A. T. Campbell, M. E. Kounavis, and R. R.-F. Liao. The mobiware toolkit: Programmable support for adaptive mobile networking. *IEEE Personal Communications - Special Edition on Adaptive Mobile Networking*, 5(4):32–34, August 1998.

[2] C. Aurrecoechea, A. T. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems Journal*, 6(3):138–151, May 1998. ACM/Springer-Verlag.

[3] A. Barto, R. Sutton, and C.W.Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):834–846, Sept 1983.

[4] R. A. Brooks. A robust layered control system for a mobile robot. Technical Report AIM-864, AI Lab, MIT, Sept. 1985.

[5] T. Edmonds. A decision engine for robot football. Technical report, Laboratory for Communications Engineering, Engineering Department, University of Cambridge, Trumpington St, Cambridge, UK, CB2 1PZ, July 1998.

[6] J. Nilsson. *Real-Time Control Systems with Delays*. PhD thesis, Lund Institute of Technology, Box 118, S-221 00, Lund, Sweden, February 1998.

[7] L. E. Parker. Local versus global control laws for cooperative agent teams. MIT AI memo no. 1357, March 1992.

[8] M. Torngren. Fundamentals of implementing real-time control applications in distributed systems. Mechatronics Lab, Dept of Machine Design, Royal Institute of Technology, S-100 44 Stockholm, Sweden, 1997.