How Elastic are Real Applications?

Rolf Neugebauer
Department of Computing Science, University of Glasgow, Scotland, U.K.
neugebar@dcs.gla.ac.uk

Abstract— **Programs are typically developed with little or no consideration of their performance under different system loads or the effect they may have on other processes competing for the same resources. To an extent, this stems from the "virtual machine" approach promoted by most mainstream operating systems. With operating systems which offer mechanisms for fine-grained control of resource allocations it becomes apparent that a central policy for allocating potentially scarce resources is not sufficient. We are currently developing a toolkit which allows programmers to systematically examine and assess the performance behaviour of a wide range of applications under different resource allocations by determining the applications' utility curves. We argue that such a toolkit is useful for the development of adaptive applications as well as for the implementation of global resource management policies. In particular, we argue that this is necessary for the application of economic models to the area of resource management, as proposed by some researchers.**

I. MOTIVATION

Resource management is a fundamental task performed by operating systems. Most current mainstream operating systems implement resource management policies which are oriented towards overall system throughput and fairness. They perform reasonably well for a mix of interactive, batch processing, or server applications. Real-time operating systems are designed to give hard guarantees on resource allocations to allow applications to perform time-critical tasks. This is achieved by allocating fixed shares of resources to processes for their entire lifetime, or at least for a relatively long duration. Multimedia operating systems (e.g., [14], [13], [6], [9], [5]) give soft real-time guarantees for resource allocations to processes which may be renegotiated dynamically at runtime, encouraging applications to adapt to the changing overall system load. This is usually achieved through explicit resource allocation and revocation. This process is commonly referred to as *QoS-Management*.

QoS-Management in operating systems has to deal with a variety of applications demanding different combinations of resources. In some systems (e.g., [6], [15]) a central resource manager is deployed to perform both system wide and application specific optimisations of resource allocations. Recently, alternative approaches have been proposed that deploy economic mechanisms for more decentralised resource management. This has been proposed for the general area of resource management in operating and distributed systems (e.g., [20], [4], [19], [8], [11]) and for multimedia operating systems in particular [18].

The general idea of these proposals is that applications are charged for resource allocations and/or usage. Prices for resources are established dynamically at runtime. It is generally assumed that prices reflect the current demand for a resource and indicate the level of congestion. Most existing work suggests the use of a bidding process to establish prices for resources (e.g., [3], [11]). Recent work in the area of pricing for communication networks suggests that prices should also reflect the marginal costs a particular resource allocation for one consumer of a resource imposes on other consumers of the same resource, e.g., [10], [7]; prices with this property are known as *shadow prices*.

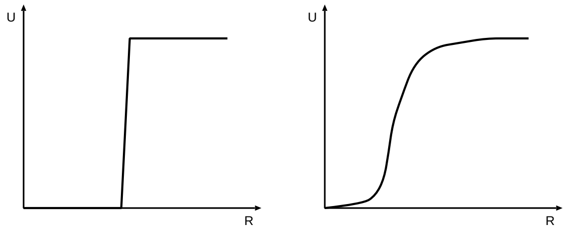

Fig. 1. Sample Utility Curves

In such an architecture, applications or groups of applications, are thought of as acting independently of each other and negotiate a resource allocation with entities managing individual resources attempting to maximise their *utility*.[1] Applications may use the feedback provided by dynamic (shadow) prices to adapt to different levels of resource congestion. Applications can facilitate this task by implicitly or explicitly modelling a *utility function*, defining the dependency of their utility on different resource allocations. These are particularly useful to application developers implementing application specific optimisation strategies.

Consider the two idealised utility curves shown in Figure 1.[2] The utility is plotted on the Y-axis while the quantity of a resource is plotted on the X-axis. The application shown on the left provides maximum utility if it receives a certain quantity of the resource and no utility at all if not. Applications with this sharp loss of utility are usually called *real-time* applications – they do not provide any scope for adaptation. The application on the right offers a different level of utility depending on the resources allocated to it. Above a certain level, the marginal utility of additional resources is slight. The same applies to very low resource allocations. In between the marginal utility of additional resources is very high. Such applications are more tolerant to the resources they are allocated, and can adapt within a range of resource allocations without incurring a disproportionately high penalty on their utility. Such applications are often called *elastic*.

Whilst resource allocators should not have explicit knowledge about applications' utility functions, it is useful, if not necessary, to make some assumptions about their general *shape* when de-

Rolf Neugebauer is funded by a Marie Curie Fellowship from the European Union, Contract-No.: ERBFMBICT972363. This work is also supported by the Pegasus II project (ESPRIT LTR 21917).

[1] In economics, utility is a measure of the usefulness of a particular quantity of a resource or a bundle of resources to individual consumers.
[2] Taken from [17].

signing a pricing mechanism for resources. Like applications, resource allocators seek to optimise the global resource allocation either to maximise their "profit" or to achieve a "social optimum". Knowledge about the general shape of applications' utility functions allows system designers to reflect on the success of these aims. To establish shadow prices, assumptions about the shape of the utility functions are necessary, since shadow prices, to an extent, reflect the marginal costs other applications have to pay for the increase of a resource allocation for one application. Note that the majority of proposals to use shadow prices for resource management assume elastic applications.

The importance of the shape of utility functions both for the applications and the overall resource allocation mechanism have led us to conduct a series of experiments to investigate this matter systematically. For this purpose, we are developing a toolkit which allows us to examine the effect of different resource allocations on the performance of a wide range of applications. The toolkit can also be used by application developers to study the effectiveness of different application specific optimisations and adaption techniques.

The primary aim of this research is to study the *shapes* of utility curves and their dependencies on different resources for a variety of real applications in order to develop a general pricing scheme for resources in operating systems. In particular, we are interested in finding out how elastic real applications are and what makes applications elastic. We are not interested in the measurement of absolute performance or quantitative comparisons of different applications since these are both implementation and operating system dependent.

In the next section we describe the architecture of our toolkit and report on a sample implementation. We give an example use of the toolkit in section III and in section IV we present our conclusions.

II. Architecture Overview

The main feature of the architecture is that an external control process systematically alters the resources allocated to a test application. This allows systematic rather than empiric experiments and requires little or no alteration of the test applications. The control process monitors the utility of the test application either by communicating with the test application or by measuring its performance externally.

Measuring the performance externally is limited to recording completion times for particular tasks or response time to requests. This is useful for measuring the performance of batch processing and server applications but is restrictive in that it is difficult to measure some application domain specific quantities, e.g., the frame rate or jitter of a video application. When such measures are important, we require test applications to provide a minimal control interface to the control process. This interface permits both synchronous and asynchronous communication.

In the synchronous case, the control process sets the resource allocation for the test application and then sends it a `start` message. After a period of time, the control application sends a `stop` message to the test application which responds with its utility. When asynchronous control is used, the control application sends a `start` message to the test application and awaits a `finished` message (including the utility) on a callback. By systematically altering the resource allocations and repeating the `start`/`stop` or `start`/`finished` sequences, the necessary information to generate utility curves is collected by the control process.

This raises two questions: what is the utility, and how can applications measure it? Unfortunately, the answer is simple but unsatisfactory: "It depends." The performance of applications can be measured based on some form of QoS metric, e.g., the ones presented in [16], [13]. For multimedia applications this is usually the number of deadlines met; batch processing applications are usually evaluated in terms of completion time or the number of requests processed and the performance of interactive applications can be measured in terms of their response time. However, utility ultimately denotes *usefulness* to the user and can be interpreted as the users' willingness to pay for a specific quantity of a resource. Consider, for example, a video display application. Whilst occasional losses of frames or jitter might be acceptable for a surveillance monitor application, it is certainly unacceptable for a high quality movie playback. Similarly, for batch processing applications, a result might not be of any use to the user if a particular deadline is missed.

We argue that these *subjective* user utility measures are dependent on *objective* utility measures as provided by a systematic performance analysis. If an application exhibits real-time characteristics (as the one depicted in figure 1) then it is unrealistic for the user to have a utility curve corresponding to a non-real-time application. If an application has an elastic or linear utility curve then different user specific utility curves can be mapped onto them by the application. The toolkit presented in this paper is only concerned with providing objective application utility curves based on measurable quantities. We envisage, however, that applications receive feedback from the user at runtime, e.g., through a credits mechanism [18] or by allowing the user to communicate his or her utility function to the application [12], which allows applications to map subjective utility curves to application utility curves at run-time attempting to maximise the user's utility.

A. Implementation

The above architecture has some requirements on the resource management as implemented by the operating system. The operating system has to offer fine grain control over resource allocations, manageable by the controlling applications. Furthermore, resource consumption must be accountable to individual processes, i.e., if servers perform tasks on behalf of a test application, then the resources consumed by the server should be accounted to the test application.

The toolkit is currently implemented on Nemesis [9] which fulfils these requirements with respect to all resources managed by an operating system. The communication between the control process and test applications is implemented using Nemesis' Inter Domain Communication (IDC) mechanism. The control application currently only alters the CPU resource allocations but we are planning to extend this to control other temporal resources such as access to the disk and the network interface. In Nemesis, both offer a similar abstraction to their internal scheduling as the CPU scheduler [1], [2], so this extension should be relatively straightforward. We also have plans

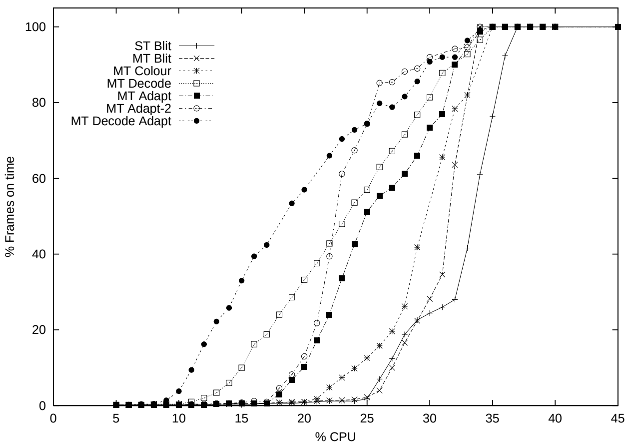

Fig. 2. JPEG Decoder Utility Curves

to extend our architecture to incorporate support for spatial resources; in particular we are interested in the performance behaviour of applications depending on the physical main memory available to them.

We are currently investigating the possibility of applying our architecture to a mainstream operating system. In particular, we are planning to perform some initial experiments using KURT[3], a real-time extension to Linux, which appears to offer a suitable infrastructure to allow fine grain control over CPU resource allocations.

III. Example

We are currently deploying the toolkit to study the utility curves of a number of applications. In this section we present our results for a particular application, a Motion JPEG video application[4]. This application reads a series of JPEG encoded frames from disk, decodes them, and displays them on the screen. The application can be configured to adapt to different resource allocations in different ways to maximise its performance (see below). It was relatively easy to fit this application into the architecture described in the last section. In the existing source code only 15 lines in one file needed to be changed. The utility measurement code can be conditionally enabled at compile time. The communication via the control interface was implemented in a separate file. The application uses asynchronous communication – after being started it attempts to display a configurable number of frames (500 for all experiments described here) before reporting back its utility to the control application.

As our measure of utility we define the number of frames which are displayed on time. For the purpose of this experiment "on time" means that frames have to be displayed in fixed intervals, corresponding to the video stream's frame rate. We allow for a variation of the display time for each frame of half of the interframe rate in each direction from the ideal display time, thus limiting jitter. The results are shown in figure 2.

The right most graph, labelled *ST Blit*, forms the base case in which no adaptation is performed. A single thread is reading data from the disk and decodes each frame. A decoded frame is then only copied to the frame buffer if it is on time. This configuration essentially forms a real-time application. For the second graph, labelled *MT Blit*, the same configuration is used, but decoding and displaying are decoupled and executed in separate threads. For the third graph the final step of the decoding process (colour conversion from YUV to RGB colour space) is moved into the display thread and dropped if a frame is not on time. For the remaining graphs this is the default mode of operation. For the fourth graph, *MT Decode*, the decoder does not decode a frame whenever the previous frame could not be displayed on time. The marginal increase in utility in the range from 10% to 30% of the CPU for this configuration is almost linear. For the next two graphs, we reduced the quality[5] of the decoding step, resulting in a less detailed image, rather than dropping frames before the decoding process. For the graph labelled *MT Adapt* the quality level is decremented by a fixed amount for every frame not displayed on time and incremented by the same amount for every frame on time. For the second graph, *MT Adapt-2*, the quality is decreased in larger decrements on every frame not on time, then it is increased for each frame on time. For the final graph, the adaptation strategies for the graphs labelled *MT Adapt-2* and *MT Decode* were combined – for every missed deadline the quality of the decoding was reduced *and* the next frame was not decoded. The resulting utility curve indicates that the video application in this configuration is fairly adaptive and can be termed elastic.

This example demonstrates how the toolkit can assist application developers to evaluate different adaption techniques for applications in order to make them elastic to different resource allocations. The information obtained from these experiments can be used to implement different strategies to react to both different users preferences and varying resource availability.

IV. Conclusion

We presented a toolkit which allows the systematic study of applications' utility functions. We argued that such a toolkit is useful to programmers developing adaptive applications allowing them to evaluate the applications' performance under different system loads. We demonstrated with a sample application that our toolkit can assist application developers to make applications elastic – a desirable property for adaptive applications.

We argued that the shape of applications' utility functions plays an important role in the design of resource pricing schemes especially if shadow prices are used. We are currently conducting a series of experiments including multimedia and batch processing applications to provide a systematic study of applications' utility functions.

Currently, the toolkit only provides support for studying the effect of CPU resource allocation on applications' utility. We are planning to extend the toolkit to include support for other temporal resources and spatial resources.

Acknowledgement

The author wishes to thank Derek McAuley, Richard Black, and Huw Evans for their fruitful discussions and constructive

[3] http://hegel.ittc.ukans.edu/projects/kurt/

[4] Originally written by Neil Stratford at the Computer Laboratory, University of Cambridge, U.K.

[5] By setting coefficients passed into the de-quantiser below a varying threshold to zero.

feedback on earlier versions of this paper. I would also like to thank the anonymous reviewers for their comments.

REFERENCES

[1] P. R. Barham. A fresh approach to File System Quality of Service. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, St. Louis, USA, May 1997.

[2] R. Black, P. Barham, A. Donnelly, and N. Stratford. Protocol Implementation in a Vertically Structured Operating System. In *Proceedings of the 22nd Annual Conference on Local Computer Networks (LCN'97)*, pages 179–188, Nov. 1997.

[3] N. R. Bogan. Economic Allocation of Computation Time with Computational Markets. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, May 1994.

[4] S. Clearwater, editor. *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[5] G. Coulson, G. Blair, P. Robin, and D. Shepherd. Supporting Continuous Media Applications in a Micro-Kernel Environment. In O. Spaniol, editor, *Architecture and Protocols for High-Speed Networks*. Kluwer Academic Publishers, 1994.

[6] M. B. Jones, P. J. Leach, R. Draves, and J. S. Barrera. Modular Real-Time Resource Management in the Rialto Operating System. In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*, May 1995.

[7] F. P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[8] J. Kurose and R. Simha. A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems. *IEEE Transactions on Computers*, 38(5):705–717, May 1989.

[9] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas In Communications*, 14(7):1280–1297, September 1996.

[10] J. K. MacKie-Mason and H. R. Varian. Pricing Congestable Network Resources. http://www-personal.umich.edu/ jmm/papers/gep.pdf, 1994.

[11] M. S. Miller and K. E. Drexler. Incentive Engineering for Computation Resource Management. In B. A. Huberman, editor, *The ecology of Computation*, pages 231–266. North-Holland, Amsterdam, Netherlands, 1988.

[12] M. S. Miller, D. Krieger, N. Hardy, C. Hibbert, and E. D. Tribble. An Automated Auction in ATM Network Bandwidth. In Clearwater [4], pages 96–125.

[13] J. Nieh and M. S. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. In *Proceedings of the 16th ACM SIGOPS Symposium on Operating Systems Principles, Operating Systems Review*, pages 184–197, Saint-Malo, France, October 1997.

[14] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource Kernels: A Resource-Centric Approach to Real-Time Systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[15] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A Resource Allocation Model for QoS Management. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[16] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence. Taxonomy for QoS Specifications. In *Proceedings of Workshop on Object-oriented Real-time Dependable Systems (WORDS 97)*, Newport Beach, CA, USA, Feb. 1997.

[17] S. Shenker. Fundamental Design Issues for the Future Internet. *IEEE Journal on Selected Areas In Communications*, 13(7):1176–1188, September 1995.

[18] N. Stratford and R. Mortier. An Economic Approach to Adaptive Resource Management. In *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, AZ, USA, Mar. 1999.

[19] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and S. Stornetta. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, February 1992.

[20] W. Walsh, M. Wellman, P. Wurman, and J. MacKie-Mason. Some Economics of Market-Based Distributed Scheduling. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, Amsterdam, Netherlands, 1998.