Economic Theory for Memory Management Optimization

(Position Paper)

Jeremy Singer
University of Glasgow, UK
jeremy.singer@glasgow.ac.uk

Richard E. Jones
University of Kent, UK
r.e.jones@kent.ac.uk

ABSTRACT

In this position paper, we examine how economic theory can be applied to memory management. We observe the correspondence between the economic notion of a consumer and an instance of a virtual machine running a single program in an isolated heap. Economic resource consumption corresponds to the virtual machine requesting and receiving increased amounts of heap memory from the underlying operating system. As more memory is allocated to a virtual machine's heap, there is additional benefit (cf. economic utility) from the extra resource. We also discuss production and cost functions, which might assist in efficient memory allocation between multiple virtual machines that are competing for a fixed amount of shared system memory.

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors—*Memory management (garbage collection)*

General Terms

Economics, Theory

Keywords

Microeconomics, Memory management, Garbage collection, Virtual machines, Resource allocation

1. INTRODUCTION

One popular definition of economics is that it deals with the *allocation of scarce resources among alternative uses* [9]. We have previously shown that microeconomic theory is effective in tuning the heap size of a garbage collected program [11] but we feel that it may have wider applications in the area of memory management for computer systems. In a multi-tasking environment, various processes are competing for shared resources. One of the fundamental commodities that can be dynamically divided between the processes is system memory.

ICOOOLPS'11, July 26, 2011, Lancaster, UK.

In this paper, we restrict our consideration to virtual machines (VMs). Each VM has its own distinct, garbage-collected heap. A single managed program executes within each VM instance. Each VM/program has a minimum heap size threshold below which program execution fails due to insufficient memory. If a VM is allocated more heap space, then less garbage collection (GC) takes place, so the program runs faster. In general, the more memory we assign to a VM, the lower the GC overhead, the faster the program executes.

Microeconomics offers a rich theory to account for the behaviour of both consumers and producers. The concept of economic *utility theory* involves measuring or predicting how much benefit a consumer gains from additional resource consumption. *Production functions* measure the change in output due to a change in the quantity of some input. *Cost functions* can be used to minimise the cost of the inputs subject to some constraint. In this work, we consider how these theories can be applied concretely to the area of memory allocation for multiple VMs. We posit that this theory can provide a principled approach to selecting which VM will gain the most benefit from receiving available free memory, or which VM will be least affected when memory must be taken from an existing process (e.g. due to system paging).

2. MOTIVATION

We envisage a system where multiple VM instances compete for memory, and there is scope for substantive redistribution of memory between VMs. A likely environment is a large shared-memory server that executes multiple jobs simultaneously. Each job is a single VM instance with an associated heap. Examples include:

1. an enterprise Java server, running multiple Java applications, each in its own isolated JVM instance.

2. a multi-core Hadoop system, with each worker node executing in its own isolated JVM instance [7].

3. a Xen server [3] running virtualized OS instances in a compute-cloud.

In such systems, the underlying operating system or hypervisor must determine how to apportion the available physical memory resources to the different VMs. There may be different policies governing the distribution of memory. All VMs may be given equal treatment, or some may have specific priorities. In a cloud computing scenario, there may be service level agreements to maintain.

In this paper, we present microeconomic apparatus to handle this kind of runtime resource allocation problem. One common criticism of memory management is that it uses ad-hoc techniques, low-level heuristics, and bottom-up engineering practice. Our research goal is to phrase memory management problems in a high-level manner that is suitable for the application of standard economic theory, in order to derive viable solutions from a principled theoretical framework.

Server configuration is a significant problem for systems administrators. It is difficult since there are so many configuration options to tune, at a variety of levels in the execution stack. OS and VM configuration often requires a heuristic approach, or some automated tuning. We have a particular interest in auto-tuning approaches to memory management, e.g. [1, 10]. Our newly adopted, alternative philosophy is to adopt a more abstract view, based on economic theory, that allows higher-level tuning.

3. MICROECONOMIC THEORY

This section outlines the basics of standard microeconomic theory in Section 3.1. Then we show how this might be applied to memory management for concurrently executing virtual machine instances in Section 3.2.

3.1 Basic Theory

Microeconomic theory allows modelling from the viewpoint of either the consumer or the producer. We believe that both viewpoints can provide a more principled framework for policy decisions in a memory manager.

3.1.1 Utility theory

Consumers desire a commodity because of its ability to satisfy their want. This want-satisfaction property is known as *economic utility*. As an individual consumer consumes more of a commodity per time period, his/her satisfaction (known as *total utility*) increases. *Marginal utility* is the extra satisfaction derived from consuming one additional unit of the particular commodity per unit of time, while holding all other consumption constant. More mathematically, marginal utility is the *derivative* of total utility.

In general, total utility increases with each additional unit of commodity. However it generally increases by smaller amounts as more and more of the commodity is consumed, i.e. the marginal utility declines with increasing units of commodity. For example, consider the marginal utility that results from acquiring one additional penny. For someone who only has one penny initially, an extra penny is significant. However for a millionaire, obtaining one more penny is negligible. In this case, the marginal utility caused by acquiring an extra penny decreases as wealth increases.

In some cases of excessive consumption, total utility reaches a saturation point and then begins to decline. This is the point at which the corresponding marginal utility becomes negative. For example, consider a personal-health-based utility function for hamburger consumption. For some value of $N > 0$, there are detrimental health effects associated with a daily intake of N hamburgers. For other commodities, the utility function is *non-satiable*. i.e. the marginal utility is always positive.[1]

[1]One might expect the utility function for memory allocation to be non-satiable, i.e. increasing the heap size of a program does not degrade its performance. However, when the heap is extremely overprovisioned, system effects like paging can degrade performance.

The *law of diminishing marginal utility* means that the first unit of consumption of a good or service yields more utility than the second and subsequent units. This is reflected in the negative slopes of marginal utility curves.

One key issue concerns how to measure values of utility. In economics, utility is a quantitative summary of consumer satisfaction or happiness. Sometimes such values are difficult to assess directly. It may be easier to use proxy measures for utility.

3.1.2 Production functions

A production function specifies the output of a firm, industry or economy for all combinations of inputs. Unlike a utility function, a production function is not based on consumer choice but on external factors such as technology or labour productivity. Production functions can be used to measure the effect on output of varying the quantity of one or more inputs. In practice, some inputs may be fairly fixed or at least 'lumpy' (for example, it is expensive to open a new factory but the gains from doing so are large) whereas other inputs (such as the number of workers) may be more flexible.

Economists are interested in MP_k, the *marginal physical product of an input* k. This is the change in output obtained from increasing the quantity of input k, or mathematically the partial derivative of the production function with respect to the quantity of input k. If inputs are substitutable (for example, the amounts of physical memory given to different VMs), then we are also interested in the *marginal rate of technical substitution (RTS)*: the rate at which one input can be substituted for another while holding the output constant.

3.1.3 Cost functions

A cost function measures the cost of production of an output as a function of the inputs and their costs. Typically, we want to minimise the total costs given a constraint on the inputs: a constrained minimisation problem. We can define the *marginal productivity per unit spent* as the ratio of the marginal physical product of an input and cost of that input. The general solution to the cost minimisation problem reveals that, to minimise costs, the marginal productivity per unit spent should be the same for all inputs.

3.2 Application to Virtual Machines

In this section, we consider how to apply the notions of microeconomic theory to memory management for virtual machines (VMs).

3.2.1 Consumers and Producers

We can consider the analogy between economics and memory management either in terms of consumers and commodities, or in terms of inputs and outputs.

- The **consumers** are individual **VM instances**. Each VM is competing with the other VMs for allocation of memory, in the system. The **commodity** is system **memory**, units of which can be allocated to a single VM instance to form part of that VM's heap.

- A **producer** transforms inputs to outputs. We consider the whole system to be the producer. The **inputs** are the units of system **memory** allocated to each VM instance. The **output** might be the total volume of bytes allocated by all programs, which is clearly constant overall (so we have an isoquant[2]).

3.2.2 Utility

The measurement of utility is somehow related to reduced garbage collection (GC) overhead. When an individual VM has a larger heap, it performs less GC. This allows the overall execution of the particular program to proceed faster.

The first suggestion is that total utility might be inversely proportional to overall *execution time spent in GC*. We could compute total utility, over the full execution of the program, as:

$$TU = 100 - \%\text{GC time} \quad (1)$$

This is similar to our earlier work on allocation curves [11]. There we considered the number of GCs incurred throughout program execution, at each VM heap size. However we suppose that there is a high correlation between the number of GCs and the time spent in GC. Using this measure for total utility, the marginal utility then becomes the derivative of the allocation curve, which is equivalent to the metric we defined in [11] as *allocation elasticity*.

A second suggestion is that the total utility might be directly proportional to the *allocation rate* of a program. A program is likely to require increased heap memory (or increased GC) if it has a high allocation rate. On the other hand, if the program is not allocating new data, then it does not require extra heap memory.

3.2.3 Cost

Whereas the previous two suggestions considered VM behaviour (time spent in GC) and mutator behaviour (allocation rate) in relative isolation, our final suggestion is a combined measure of both GC and allocation activity. This is the *mark/cons ratio*, which is a standard GC metric, e.g. [4]. *Mark* is a measure of GC activity (how many memory cells are marked as live during a single GC trace). *Cons* is a measure of mutator activity (how many new memory cells are allocated on the VM heap during program execution between two consecutive GCs). When the mark/cons ratio is high, there is a lot of GC in relation to allocation. When the mark/cons ratio is low, there is a high amount of allocation in relation to GC activity. A GC optimization is accepted as useful if it reduces the mark/cons ratio, e.g. [2, 12].

As the VM heap size increases, so the amount of GC reduces; thus the mark/cons ratio should go down. Thus we suggest that the mark/cons ratio might be another potential proxy measurement for *marginal utility* for memory management. The law of diminishing marginal utility is satisfied empirically—in general, further heap increases gives smaller reductions in the mark/cons ratio.

However a further option is to use the mark/cons ratio as the *productivity function* for a VM instance. In fact, to get the mathematics correct, we need to use the inverse of the mark/cons ratio. This inverse ratio will increase as the amount of GC decreases, i.e. the productivity is correlated with the proportion of useful work performed by the VM.

One factor in favour of the mark/cons ratio is that it is already a well-known and accepted metric in the GC community. One advantage of considering the mark/cons ratios of each executing program is that it takes into account both the *rate* at which a program is allocating memory and its *volume of live memory*. For example, it is likely to be desirable to differentiate in terms of memory resources given to a program that is both allocating and freeing rapidly, and one which has a much more steady-state behaviour. It may therefore be better to phrase production or cost functions in terms of mark/cons ratios.

[2]An isoquant is a contour line running through a set of points at which the same output is produced while varying the quantities of the inputs.

4. DIRECTIONS FOR FUTURE RESEARCH

Our overall goal for this research project is to apply economic theory in a multi-VM environment, to determine how to share memory effectively between the VM instances. As the system load varies dynamically, as new processes start and existing processes complete execution, the OS has to co-operatively resize heaps to allocate memory effectively to the multiple VMs, whilst avoiding paging due to excessive memory allocation. There are existing research systems that do this (e.g. [5, 15, 14]) but none that work co-operatively with all processes, and none that use economic theory to determine their outcome.

Some questions remain, which will influence the system design. There are issues about prioritization of VMs. Initially we will assume that they all have equal priority. However we can imagine situations where users set priorities for different VMs, or there are service-level agreements to maintain which require VM prioritization.

Another question involves the measurement of utility, production or cost curves. Given suitable profiling runs, it might be possible for each Java program to have a representative utility curve embedded in its meta-data. However there are weaknesses to this approach. We are aware that Java program GC behaviour can be extremely input-dependent [8] or that the meta-data could be modified to give particular programs unfair advantages. Another possibility is to keep a running measure of utility at runtime, computed over a window of recent activity. This dynamic profiling approach may incur extra overhead, but such statistics-gathering techniques are common in adaptive runtime systems.

5. RELATED WORK

As already mentioned, we have explored some of the analogies between microeconomics and memory management in our earlier work [11]. We introduced *allocation curves* as an analogue of *demand curves*. An allocation curve plots the number of GCs incurred during program execution against the fixed VM heap size used for that execution. As remarked in Section 3.2, this could give rise to a possible measure of utility. Further work in [11] controls heap growth based on the allocation elasticity metric, which is computed from changes in the heap size and the GC activity.

Hertz et al [6] present a heuristic-based approach to optimize memory management for multiple VMs in a shared memory environment. Vengerov [13] presents an analytical model for reducing the time a single VM spends in GC, based on the relative sizes of the young and old generations

in the heap, and the promotion criterion. Again, this is a heuristic-based technique. We hope to achieve similar goals using an application of economic theory.

6. CONCLUDING REMARKS

In this position paper, we have proposed that there is a rich vein of economic theory that we can apply to memory management.

This includes concepts such as:

- utility functions 3.1.1
- production functions 3.1.2
- cost functions 3.1.3
- supply and demand [11]

We argue that economic theory gives a broad selection of tools to correlate inputs and outputs, supply and demand, minimising cost etc. We aim to apply these tools to the domain of memory management or, at least, use them as a source of inspiration.

7. REFERENCES

[1] E. Andreasson, F. Hoffman, and O. Lindholm. To collect or not to collect? machine learning for memory management. In *Proceeedings of the 2nd Java Virtual Machine Research and Technology Symposium*, pages 27–39, 2002.

[2] H. G. Baker. Infant mortality and generational garbage collection. *SIGPLAN Notices*, 28:55–57, April 1993.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 164–177, 2003.

[4] W. D. Clinger and F. V. Rojas. Linear combinations of radioactive decay models for generational garbage collection. *Science of Computer Programming*, 62(2):184–203, 2006.

[5] M. Hertz, Y. Feng, and E. D. Berger. Garbage collection without paging. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 143–153, 2005.

[6] M. Hertz, S. Kane, E. Keudel, T. Bai, C. Ding, J. Bard, and X. Gu. Waste not, want not: Resource-based garbage collection. In *Proceedings of the 2011 International Symposium on Memory Management*, 2011. *(to appear)*.

[7] S. Ibrahim, H. Jin, L. Lu, L. Qi, S. Wu, and X. Shi. Evaluating mapreduce on virtual machines: The hadoop case. In *Cloud Computing*, volume 5931 of *Lecture Notes in Computer Science*, pages 519–528, 2009.

[8] F. Mao, E. Z. Zhang, and X. Shen. Influence of program inputs on the selection of garbage collectors. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 91–100, 2009.

[9] D. Salvatore. *Microeconomics theory and applications*. Oxford University Press, 5th edition, 2008.

[10] J. Singer, G. Brown, I. Watson, and J. Cavazos. Intelligent selection of application-specific garbage collectors. In *Proceedings of the 6th International Symposium on Memory Management*, pages 91–102, Oct 2007.

[11] J. Singer, R. E. Jones, G. Brown, and M. Luján. The economics of garbage collection. In *Proceedings of the 2010 International Symposium on Memory Management*, pages 103–112, 2010.

[12] D. Stefanović, M. Hertz, S. M. Blackburn, K. S. McKinley, and J. E. B. Moss. Older-first garbage collection in practice: evaluation in a Java virtual machine. In *Proceedings of the 2002 Workshop on Memory System Performance*, pages 25–36, 2002.

[13] D. Vengerov. Modeling, analysis and throughput optimization of a generational garbage collector. In *Proceedings of the 2009 International Symposium on Memory Management*, pages 1–9, 2009.

[14] T. Yang, E. D. Berger, S. F. Kaplan, and J. E. B. Moss. Cramm: virtual memory support for garbage-collected applications. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, pages 103–116, 2006.

[15] C. Zhang, K. Kelsey, X. Shen, C. Ding, M. Hertz, and M. Ogihara. Program-level adaptive memory management. In *Proceedings of the 5th International Symposium on Memory Management*, pages 174–183, 2006.