Virtual Private Machines: User-Centric Performance

David Bartholomew Stewart and Richard Mortier
Microsoft Research Ltd., Cambridge, UK

Abstract

Inconsistent system behavior causes unpredictable performance which is known to stress users; making the system perform consistently should remove this source of user stress. Operating systems currently provide the illusion that each application runs on a dedicated Virtual Machine*. This paper proposes incorporating performance into this abstraction, resulting in a* Virtual Private Machine*. The VPM abstraction aims to improve user-perceived performance by increasing performance consistency, and it is applicable to any user-visible application, from word processors to web servers. To provide VPMs, per-resource performance models allow resources to be scheduled to meet target response times calculated for each user-visible action.*

1. Introduction

Inconsistent computer system performance is known to be a significant contributor to poor user experience by causing frustration [1] and stress [2]. Users base their impression of system performance upon their applications' response times: given consistent response times, users can form clear expectations of system performance. Conversely, variable response times prevent users forming such well-defined expectations, resulting in surprise when the system performs quickly, frustration when the system responds slowly, and generally a state of confusion about what the performance of the system actually is and whether it improves or degrades over time.

We believe that even small amounts of variability can subconsciously affect users. We therefore claim that making system behavior *more consistent*, even if not perfectly consistent, will improve the user experience: a small degradation in absolute system performance is worth paying to achieve greater predictability.

To this end, we propose the abstraction of a *Virtual Private Machine (VPM)*, applicable to all types of computer system with user-perceptible performance, from individual desktop machines supporting single-user applications such as word processing, to clusters of machines supporting multi-user applications like databases. The goal of this abstraction is to provide more consistent service with only minimal performance degradation.

Traditional operating systems provide *virtual machines* which virtualize system *functionality* to give users the illusion that each application runs on a dedicated system. A Virtual *Private* Machine is provided by a *User-Centric Resource Manger (UCRM)* by extending this illusion to include *performance* by managing system resources to allow each application to perform as if alone.

Throughout this paper *action* refers to some higher level user-application initiated system activity, and *system demand* refers to the set of *resource demands* made to resources in the system. For example, an action might refer to the user requesting a word processor to save a document; this generates a set of system demands such as "open", "read", "write", each of which generate resource demands such as disk reads and writes, bursts of computation, and network activity.

There are then three principal functions that a UCRM must provide to implement a VPM:

1. It must infer actions by observation of application and system behavior.
2. It must then calculate response time targets for the system demands generated by an action.
3. It must finally schedule resources to meet these targets.

If the system is not overloaded the intended result is more consistent system behavior leading to more predictable application performance, thus removing one source of user stress.

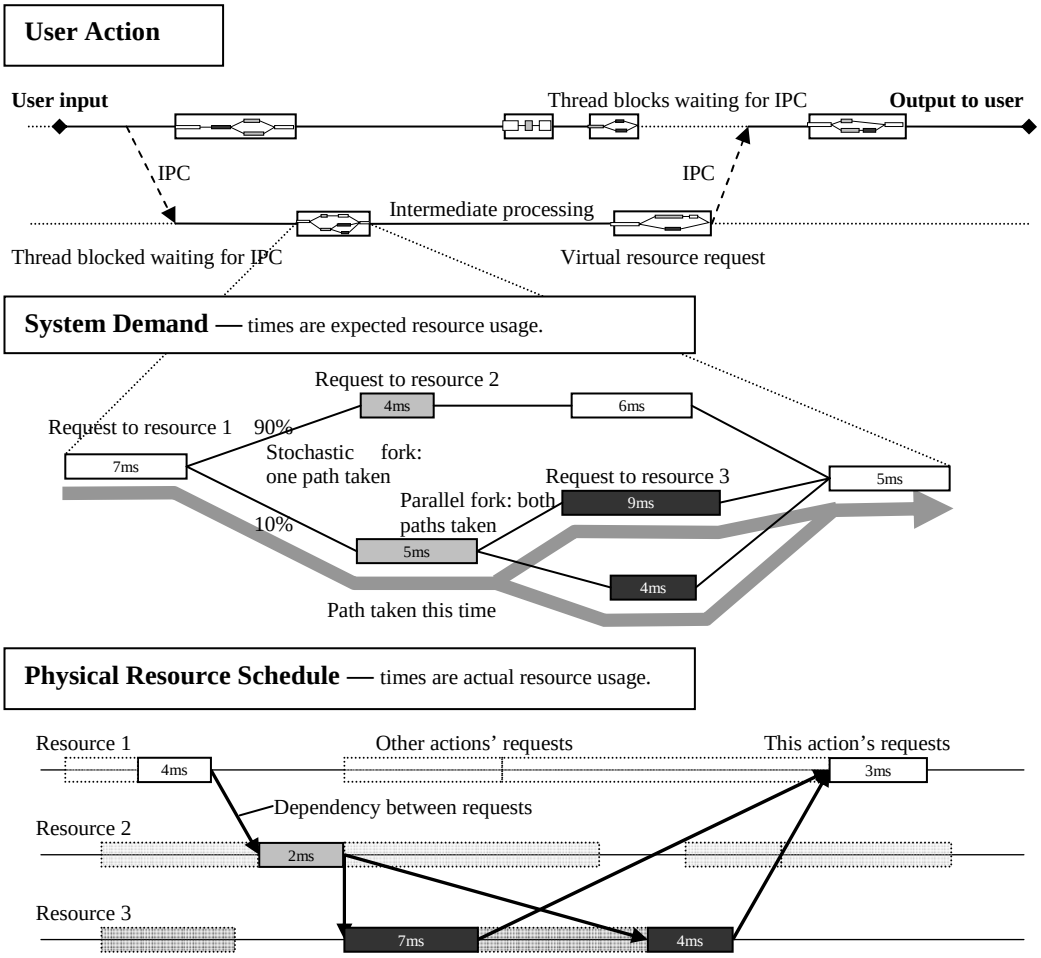

Figure 1. Example user action comprising 6 system demands, with detail given for one system demand, and a particular instance of a resource schedule generated to service that demand.

Figure 1 illustrates the relationship between actions, system demands and resource demands as time progresses from left to right. From the top, the figure shows a single multi-threaded user action making six system demands, 4 associated with the immediate application's process and 2 associated with another process. It then shows the model of resource usage for a particular system demand, in this case one that uses three physical resources. The resource usage model is represented as a stochastic finite state machine with nodes annotated with the expected resource usage times, and links annotated with the likelihood that the link is followed. The thick gray line shows the path taken in this particular instance. Finally, the bottom part of the figure shows the physical resource requests being scheduled on the actual devices where there is contention from other user actions.

The following section considers related work, and Sections 3—5 deal with the three functions of the UCRM noted above. Section 6 describes initial implementation of a UCRM disk scheduler and Section 7 concludes.

2. Related Work

The problem of accounting resource consumption to some higher-level entity is actively being addressed. Magpie [3] attempt to assign resource consumption to higher level "requests" based on an event stream generated by the application and the system at run-time. Resource Containers [4] allow the system to rebind a thread to different "resource containers" based on the thread's behavior.

There is also a large body of work aimed at resource scheduling for "better service" [11][12]. However, the meaning of "better service" varies: for example, Nemesis [11] provides throughput guarantees to ensure applications perform as well as possible given available resources. Also, these studies treat resources independently: an under-loaded resource cannot be scheduled to make up for an overloaded resource.

Finally, previous studies have principally considered low-level resource scheduling. Scheduling algorithms designed to minimize lateness exist [5][6], but they have not been applied in the context of user actions. Also, OS services such as buffer caching introduce significant performance variability for applications.

3. Accounting

Resource demands must be accounted to user-initiated actions in order to schedule resources to provide consistent performance. Since modifying all applications is not feasible, the UCRM should account resource demands to user actions using only application behavior and the system interaction with the user.

Two simplifying assumptions aid action identification: (*i*) threads service only one action at a time; (*ii*) the beginning and end of an action are marked by input and output respectively. These assumptions may be broken, by application-private caches for example. As Section 2 notes, we leave the general problem of ascribing low-level resource demands to high-level user actions to be addressed by related work. System demands may then be linked to an action by observing inter-thread communication. Physical resource demands can be tied to system demands using explicit accounting mechanisms, avoiding reliance on further inference from user and application behavior.

An action can thus be viewed as a partially ordered set of system demands, modeling asynchrony and multi-threaded actions. Each system demand is then modeled as a partially ordered set of physical resource demands which model performance including caching and other system artifacts.

4. Target Calculation

After inferring an action, the UCRM must set response time targets for the system demands generated by the action. The aim is that targets will be set so that all instances of the action will have similar response times. These targets will then be translated into completion time targets for the physical resources in the system.

There are two challenges when calculating these targets: (*i*) there are too many types of user action simply to list each with its duration, and in any case, new applications would only require extending this list; (*ii*) translating action targets to system demand targets requires knowing in advance the system demands an action will generate, and how long each will take.

Rather than maintain such a list we impose the system demand abstraction at a sufficiently high level that each type of action will generate the same system demands. Response time targets for a system demand can then be used to calculate target completion times for the demands it places on physical resources. For example, a target completion time for system demands that might access a disk are independent of whether the particular blocks being retrieved are currently cached.

Completion time targets for a system demand are calculated using their physical resource usage model defined by the operating system, the expected contention for the physical resources from other actions set by the system administrator, and the actual contention due to this action making multiple demands on the same physical resource. Contention arising within an action reflects the cost of multiple outstanding physical resource requests; targets will still be consistent since this contention is consistent between different instances of a given type of action.

Finally, physical resource schedulers can use these completion time targets in conjunction with physical resource usage models to generate a schedule. Meeting these schedules will ensure consistent performance. A physical resource usage model is a stochastic finite state machine where each state is a resource demand annotated with expected response time. Link probabilities and expected response times are derived from observed system behavior. The scheduler will then use the *mean* time taken to complete a virtual resource demand under contention from other demands from that action as its target response time. Although using worst case behavior would guarantee all targets were satisfied, this would have too high a performance cost.

To illustrate, consider the action of a file save, generating a system demand to read part of the file and assume no competing demands from that action. A performance model states this system demand needs 2 ms CPU time followed by a virtual disk access, which translates to 1ms CPU time if the data is cached, or a disk access taking 15 ms otherwise. Assume the cache has a 90% hit rate, and the administrator has set contention levels of 3:1 for the CPU and 1.2:1 for the disk. The target response time of this system demand is $2\times3 + (15\times1.2\times10\% + 1\times3\times90\%) = 6 + 4.5 = 10.5$ ms. These figures are used to set the response times for the physical resource requests: the initial CPU request should be finished in 6 ms, and activity due to the subsequent disk demand or cache hit should complete 4.5 ms later.

5. Scheduling

Finally, the UCRM must schedule physical resources to meet user action targets. This can be achieved by scheduling each physical resource to meet its system demands' target completion times.

Late or early completion of one physical resource demand should not affect target completion times for subsequent demands: targets for the demands comprising a single action can be calculated as absolute times. If an action has a resource demand that cannot be serviced on time, subsequent demands of that action can be scheduled to make up for any lateness. Traces of modern applications show that a user action typically consists of tens, if not hundreds, of system demands, giving considerable scope for amortizing lateness.

Lateness that persists to the end of the action will be observable by the user. Consequently, individual resource schedulers should attempt to minimize the amount of late time that schedulers handling subsequent demands must deal with. If an action completes early, the effects can be hidden by delaying output reporting the end of the action.

The core of the resulting scheduling algorithm is as follows: the demand next serviced is that which, if not serviced immediately, would cause the greatest additional total lateness across all actions. A demand on which several actions would block is given a completion time target per action; when a target is missed, the rate at which not servicing that demand adds lateness is increased.

Specific resources require extensions to this algorithm, although only disk and network resources require only minor extensions. Disk scheduling is dealt with in the case study in the following section. For network protocols such as TCP, we claim that network packets really have two target response times: reception and transmission must be considered separately, since receipt of an acknowledgement causes data to be transmitted. The completion time target then becomes simply the earlier of the two targets.

CPU resource cannot be scheduled using this algorithm since actions do not indicate how much CPU they will require when they are scheduled. This problem can be elided by quantizing CPU resource and treating it as if it cannot be pre-empted, allowing schedulers such as periodic Earliest Deadline First (EDF) to perform better [7].

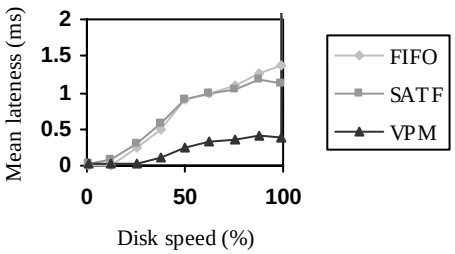

Figure 2. Disk scheduling results.

6. Case Study: Disks

Work building the UCRM has just begun; the only implemented scheduler is for the disk. This is virtualized at the buffer cache layer, so the it encompasses the various system caches as well as the physical disk.

We define *lateness* as the amount by which a resource demand misses its target completion time; if negative, it is sometimes termed *earliness*. Given that target completion times are calculated precisely to provide consistent performance, lateness in the system tends to be a measure of inconsistent performance[1]. Consequently, we use the mean lateness across all actions as the system performance metric. We treat earliness as zero rather than negative lateness since earliness also positively contributes to performance variability; earliness is easily dealt with by appropriately delaying presentation of results to the user however.

Disks perform particularly poorly if their scheduling algorithms ignore request locality. Consequently, we modify the core scheduling algorithm to use disk rotation effects and the position of the disk heads after fulfilling the previous demand when calculating the system lateness each demand will introduce. The scheduler then selects the demand that will cause most lateness, R_L. To make more efficient use of the disk, any demands that will not affect the completion time of R_L but have shorter access times than R_L are scheduled first, before finally schedules R_L.

More complex optimizations are possible [8], but even this quite simple scheduler makes a considerable difference: it substantially reduces total system lateness, without degrading total disk throughput.

Figure 2 gives simulation results obtained when using our disk scheduler latency targets

[1] If a particular action is persistently late, then this can be considered a system failure, requiring improved schedulers and/or physical resource usage models.

under a synthetic workload. This workload is based on traces of disk activity on a UNIX server with approximately 20 users. The *x*-axis is the speed of the virtual device as a percentage of the real device; the *y*-axis is the observed mean lateness averaged over 8000 actions. Results for three schedulers are given: FIFO (first-in first-out), SATF (shortest access time first) [10], and our scheduler. A simple Earliest Deadline First scheduler was also implemented, but without disk specific optimization it performs too poorly to be shown on the same scale.

The results show that with a virtualized disk running at 95% of the speed of the physical disk (i.e the contention ratio was set to 20:19), this scheduler achieves the goal of low lateness notably better than traditional schedulers. The point is not the absolute reduction in lateness (only a few milliseconds in absolute terms), but that with a single resource scheduler, lateness has been reduced by 50%. With appropriate scheduling of other resources, it should be possible to eliminate even more performance variation.

Furthermore, this synthetic workload provides relatively short actions as a result of the simple applications that generated the original traces. Modern applications tend to be more complex and to generate much longer actions, with corresponding potential for higher variability. In such cases we believe a 50% reduction in lateness would make a significant difference to the consistency of system performance.

7. Summary

We have proposed a system aimed at reducing perceived performance variability for users in order to reduce user stress. We discussed the principle components of this system and presented a brief case study of an implementation of one scheduler in the system.

Better workloads are required for further simulations comparing schemes for identifying user actions, calculating response time targets, and scheduling requires better workloads. We intend to use traces of activity gathered on real systems: we have begun collecting disk traces for the disk simulation discussed in Section 6, and will extend this to cover other resources.

The system performance models used to set target response times could also be used to cope with unexpected contention for resources. If high demand for one resource prevents some targets being met, resources under low demand can be biased toward serving those requests early to make up the deficit. This might be modeled as a market in slack time between resources, leveraging the variety of work applying economics to resource scheduling [13].

Finally, to prove this concept it is clearly necessary to augment a commodity operating system and perform a user study to show that our system actually achieves the goal of reducing user stress.

References

[1] J. Scheirer, R. Fernandez, J. Klein, R.W. Picard. *Frustrating the user on purpose: a step toward building an affective computer*. Interacting with Computers 14, 93–118, Elsevier Press, 2002.

[2] W. Boucsein, *Chapter 14 of "Engineering Psychophysiology: Issues and Applications", The use of psychophysiology for evaluating stress-strain processes in human computer interaction.* R.W. Backs and W Boucsein, editors. Lawrence Erlbaum Associates, Publishers, 2000.

[3] P.R. Barham, R. Isaacs, R.M. Mortier and D. Narayanan. *Magpie: online modeling of performance-aware systems*, In 9th Workshop on Hot Topics in Operating Systems (HotOS-IX), Lihue, Hawaii, May 2003.

[4] G. Banga, P. Druschel, and J.C. Mogul. *Resource containers: A new facility for resource management in server systems.* In Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI '99), 45–58, February 1999.

[5] L. Abeni and G. Buttazzo. *Integrating multimedia applications in hard real-time systems*. In Proceedings of The IEEE Real-Time Systems Symposium, 3–13, December 1998.

[6] J.A. Stankovic, M. Spuri, M. Di Natale, and G.C. Buttazzo. *Implications of classical scheduling results for real-time systems.* IEEE Computer, 28(6):16–25, June 1995.

[7] C. L. Liu and J. W. Layland. *Scheduling algorithm for multiprogramming in a hard-real-time environment.* Journal of the ACM, 20(1):46–61, 1973.

[8] S. Chen, J.A. Stankovic, J.F. Kurose and D. Towley. *Performance evaluation of two new disk scheduling algorithms for real-time systems.* Journal of Real-Time Systems, 3:307–336, 1991.

[9] C. Ruemmler and J. Wilkes. *HP Laboratories UNIX disk access patterns* Technical report HPL–92–152, Hewlett-Packard Company, December 1992. Also published in USENIX Winter 1993 Technical Conference.

[10] D.M. Jacobson and J. Wilkes. *Disk scheduling algorithms based on rotational position*. Technical report HPL–CSP–91–7rev1. Hewlett-Packard Company, 16 February 1991.

[11] I.M. Leslie, D. McAuley, R.J. Black, T. Roscoe, P.R. 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.

[12] R. Rajkumar, K. Juvva, A. Molano and S. Oikawa. *Resource Kernels: a resource-centric approach to real-time systems.* Proceedings of the SPIE conference on Multimedia Computing and Networking, January 1998.

[13] N.A. Stratford and R.M. Mortier. *An economic approach to adaptive resource management*. In 7th Workshop on Hot Topics in Operating Systems (HotOS-VII), pages 142–147, Rio Rico, Arizona, March 1999.