New Resource Control Issues in Shared Clusters

Position Statement

Timothy Roscoe
Sprint Advanced Technology Labs
1 Adrian Court
Burlingame, CA 94010, USA
troscoe@sprintlabs.com

Prashant Shenoy
Department of Computer Science
University of Massachusetts
Amherst, MA 01003, USA
shenoy@cs.umass.edu

Abstract—**We claim that the renting of machine resources on clusters of servers introduces new systems challenges which are different from those hitherto encountered, either in multimedia systems or cluster-based computing. We characterize the requirements for such "public computing platforms" and discuss both how the scenario differs from more traditional multimedia resource control situations, and how some ideas from multimedia systems work can be reapplied in this new context. Finally, we discuss our ongoing work building a prototype public computing platform.**

I. Introduction and Motivation

THIS paper argues that the growth of shared computing platforms poses new problems in the field of resource control that are not addressed by the current state of the art, and consequently there exist important unresolved resource control issues of interest to the multimedia systems community.

The scenario we examine in detail is that of a *public computing platform*. Such a platform consists of a cluster of processing nodes interconnected by a network of switches and provides computational resources to a large number of small third-party *service providers* who pay the provider of the platform for the resources: CPU cycles, network bandwidth, storage space, storage bandwidth, etc. The platform provider offers service providers a platform which can be, for example, highly available, managed, and located in a geographically advantageous location such as a metropolitan area. In return, the platform provider can use economies of scale to offer service hosting at an attractive rate and still generate profit.

Public computing platforms differ from current hosting solutions in that there are many more services than machines: lots of services share a relatively small number of nodes. The challenge for the platform provider is to be able to sell resources like processor cycles and predictable service to many service providers, who may be mutually antagonistic, in a cost-effective manner.

This engineering problem subsumes other important scenarios as well. One example is workgroup clusters: a cluster of compute servers shared by a workgroup or an university department. Here the basic challenges are the same, but there can be more trust between applications sharing the computing facility and users are not necessarily directly paying for computation.

There is evidence that this problem is becoming important. Systems for running one, specialized class of application (e.g. web servers, caches, some Application Service Providers) in this manner are already appearing in the marketplace. However, the lack of solutions for the more general problem has prevented the range of services offered in this way from being widened, for example to include multimedia traffic.

Two research areas feed directly in to this area: both have much to offer, but do not address areas specific to the support of time- and resource-sensitive applications on public computing platforms.

A. Resource control in multimedia systems

Resource control has been central question in multimedia systems research for at least the past 10 years or so. Control of resources within a machine is now relatively well-understood: it has been addressed in completely new operating systems (e.g. [1], [2]), modifications to existing operating systems (e.g. [3]), schedulers ([4]), and abstractions ([5]).

Many of these advances were motivated by the desire to handle multimedia and other time-sensitive applications. Such mechanisms clearly have a place in a public computing platform designed to handle a diversity of services, not simply for multimedia applications but to provide performance isolation between services owned by providers who are paying for resources. Consequently, public computing platforms enable much of the past and on-going research on resource control for multimedia systems to be applied to *a new and more general setting*. The caveat though is that most of these techniques were developed for single machine environments and do not directly general-

ize to multi-resource environments (multiprocessors, clusters), for example see [6]. Consequently we argue the need for additional research to tailor these techniques to clustered environments such as public computing platforms.

B. Cluster-based computing platforms

Much work has been performed recently on the use of clustered computing platforms for network services (see [7] for an example and convincing arguments in favor of the approach). This work aims at delivering high-capacity, scalable, highly-available applications, usually web-based.

Typically, a single application is supported, or else the applications are assumed to be mutually trusting—a reasonable assumption in the large enterprise case. Consequently, little attention is paid to resource control, either for real-time guarantees to applications or performance isolation between them [8]. Similarly, intra-cluster security is relaxed as a simplifying assumption within the platform [9].

One notable exception to this is recent work on providing differential service to web-based applications, for example Cluster Reserves [10]. This work assumes a large application running on a cluster of servers, where the aim is to provide differential service to clients based on some notion of service *class*, for example requested content or source address. Many, though by no means all, services on the Internet today fall into this category. In the future we can expect a wider variety of services with a wider range of resource requirements.

While the arguments for an approach based on clusters of commodity machines carry over into the public computing space, the assumptions about resource control and trust clearly do not: the applications we can expect to be running on such platforms will have diverse requirements and the operators of such applications will be paying money to ensure that those requirements are met. In addition, they may be in competition with each other. Lack of trust between competing applications as well as between applications and the platform provider introduces new challenges in design of cluster control systems.

C. What's different about public computing platforms

This paper argues that the systems problems of public computing platforms are conveniently similar to the two fields above, but have a specificity of their own. They both present new challenges, but also have properties that help to ground and concretize general classes of solutions.

The most significant property of systems like this that set them apart from traditional multimedia systems and cluster-based servers is that resources are being *sold*. From a cluster architecture point of view this means that performance isolation becomes central: it is essential to provide some kind of quantitative resource guarantees since this is what people are paying for.

From a multimedia systems point of view this property has two effects. Firstly, resource allocation must extend over multiple machines running a large number of services. This amounts to a problem of *placement*: which components of which services are to share a machine?

Secondly, the policies used to drive both this placement and the resource control mechanisms on the individual machines are now driven by a clear business case. Resource control research in the past has been marked by a lack of clear consensus over what is being optimized by the various mechanisms and policies: processor utilization, application predictability, application performance, etc. The notion of graceful degradation is also made more quantitative in this scenario: we can relate degradation of service to a change in platform revenue. This represents a significant advance over current so-called "economic" or "market-driven" resource allocation policies since they can now be explicitly linked to a "real" market.

We elaborate on these issues below.

II. Challenges in Designing a Public Computing Platform

We first describe the design challenges that arise from the perspective of a platform provider. We then discuss challenges that must be addressed from the perspective of platform users (i.e., service providers). Finally we discuss the implications of these challenges for system design.

A. Challenges for the Platform Provider: The Need for Overbooking and Yield Management

The primary goal for the operator of a public computing platform is to maximize revenues obtained from renting platform resources to service providers. A public computing platform services a wide variety of customers; depending on how much each customer pays for resources, not all users are treated equally. This process is known as *yield management*, and adds an important twist to the policy side of the resource control problem. Maximizing yield (revenue) requires that platform resources be overbooked. Overbooking of resources is typically based on an economic cost-benefit analyses, which explicitly links resource allocation not to closed market abstractions (e.g. [11]) but to a "real" commercial operation.

Beyond this, resource policies will take into account such factors as demographics and psychometric models of client behavior in determining allocations and pricing. In

other industries where similar challenges exist (for example, the airline industry [12]), much of this is in the domain of business decision-making and operations research models. The challenge for a public computing platform is to allow as much flexibility as possible in business decisions regarding its operation: it must not impose undue restrictions on business policies, but at the same time should facilitate their implementation.

From a systems design point of view this has a number of implications. Firstly, business assessments and policies must be representable in the system, without the system constraining this representation (in other words, without the system taking over too much of the decision-making process). Secondly the system should aid the process of overbooking and reacting to the overloads resulting from overbooking. For instance, service providers that pay more should be better isolated from overloads than others; to achieve this goal, resource control policies should help determine (i) how to map individual applications to nodes in the cluster, (ii) the amount of overbooking on each individual node depending on the yield from that node and the service guarantees that need to be provided to applications, and (iii) how to handle an overload scenario. Thirdly, the system should provide timely feedback into the business domain as the results of the process and the behavior of other commercial parties involved (principally the service providers).

B. End-User Challenges: The Need for Appropriate Abstractions

Applications running on a public computing platform will be inherently heterogeneous. One can expect such platforms to run a mix of applications such as streaming audio and video servers, real-time multiplayer game servers, vanilla web servers, and ecommerce applications. These applications have diverse performance requirements. For instance, game servers need good interactive performance and thus low average response times, ecommerce applications need high aggregate throughput (in terms of transactions per second), and streaming media servers require real-time performance guarantees. For each such application (or service), a service provider contracts with the platform provider for the desired performance requirements along various dimensions. Such requirements could include the desired reservation (or share) for each capsule as well as average response times, throughput or deadline guarantees. To effectively service such applications, the platform should support flexible abstractions that enable applications to specify the desired performance guarantees along a variety of dimensions.

We propose the abstraction of a *capsule* to express these requirements. A capsule is defined to be that component of an application that runs on an individual node; each application can have one or more capsules, but not more than one per node. It's important to note that capsules are a *post facto* abstraction: for reasons detailed in [13] we try not to mandate a programming model for service authors. Capsules are therefore an abstraction used by the platform for decomposing an existing service into resource principals.

A capsule can have a number of attributes, such as the desired CPU, network and disk reservations, memory requirements, deadline guarantees, etc., that denote the performance requirements of that capsule. Due to the commercial nature of a public computing platform, capsules are a flexible and natural abstraction for expressing the performance requirements of applications to the system and for appropriate accounting of resource usage.

C. Implications for System Design

The above research challenges have a number of implications on the design of a public computing platform. In what follows, we discuss some of these issues.

C.1 *Capsule Placement*

A typical public computing platform will consist of tens or hundreds of nodes running thousands of third-party applications. Due to the large number of nodes and applications in the system, manual mapping of capsules to nodes in the platform is infeasible. Consequently, an automated capsule placement algorithm is a critical component of any public computing platform. The aim of such an algorithm is clearly to optimize revenue from the platform, and in general this coincides with maximizing resource usage. However, a number critical factors and constrains modify this:

Firstly, the algorithm must run incrementally: services come and go, nodes fail, and are added or upgraded, and all this must occur with minimal disruption to service. This means, for instance, that introducing a new capsule must have minimal impact of the placement of existing capsules, since moving a capsule is costly and may involve violating a resource guarantee.

Secondly, capsule placement should take into account the issue of overbooking of resources to maximize yield; sophisticated statistical admission control algorithms are needed to achieve this objective. Much of the past work on statistical admission control has focussed on a single node server; extending these techniques to clustered environments is non-trivial.

Thirdly, there are technological constraints on capsule placement, for example capsules are generally tied to

a particular operating system or execution environment which may not be present on all nodes.

Finally, there are less tangible security and business-related constraints on capsule placement. For example, we might not wish to colocate capsules of rival customers on a single node. On the other hand, we might colocate a number of untrusted clients on a single node if the combined revenue from the clients is low.

To help make this last constraint tractable, and also integrate notions of overbooking, we introduce the twin abstractions of *trustworthiness* and *criticality*, explored in more detail in [14]. These concepts allow us to represent business-level assessments of risk and cost-benefit at the system level.

Trustworthiness is clearly an issue, since third-party applications will generally be untrusted and mutually antagonistic; isolating untrusted applications from one another by mapping them onto different nodes is desirable. Trustworthiness is a function of many factors outside the scope of the system (including legal and commercial considerations), but the placement of capsules must take trust relationships into account.

The complementary notion of criticality is a measure of how important a capsule or an application is to the platform provider. For example, criticality could be a function of how much the service provider is paying for application hosting. Clearly, mapping capsules of *critical* applications and *untrusted* applications to the same node is problematic, since a denial of service attack by the untrusted application can result in revenue losses for the platform provider.

In summary, capsule placement becomes a multi-dimensional constrained optimization problem—one that takes into account the trustworthiness of an application, its criticality and its performance requirements.

C.2 *Resource Control*

A public computing platform should employ resource control mechanisms to enforce performance guarantees provided to applications and their capsules. As argued earlier, these mechanisms should operate in multi-node environments, should isolate applications from one another, enforce resource reservations on a sufficiently fine timescale, and meet requirements such as deadlines. These issues are well understood within the multimedia community for single node environments. For instance, hierarchical schedulers [15] meet these requirements within a node. However, these techniques do not carry over to multi-resource (multi-node) environments. For instance, it was shown in [6] that uniprocessor proportional-share scheduling algorithms can cause starvation or unbounded unfairness when employed for multiprocessor or multi-node systems. Consequently, novel resource control techniques need to be developed to meet the performance requirements of distributed applications in public computing platforms.

C.3 *Failure Handling*

Since high availability is critical to a public computing platform, the platform should handle failures in a graceful manner. In contrast to traditional clusters, the commercial nature of a public computing platform has an important effect on how failures are handled: we can classify failures as to whose responsibility it is to handle them, the platform provider or a service provider.

We distinguish three kinds of failures in a public computing platform: (i) platform failures, (ii) application failures, and (iii) capsule failures.

A platform failure occurs when a node fails or some platform-specific software on the node fails. Interestingly, resource exhaustion on a node also constitutes a platform failure—the failure to meet performance guarantees (since resources on each node of the platform may be overbooked to extract statistical multiplexing gains, resource exhaustion caused due to the total instantaneous demand exceeding capacity results in a violation of performance guarantees). Platform failures must be dealt with by detecting them in a timely manner and recovering from them automatically (for instance, by restarting failed nodes or by offloading capsules from an overloaded node to another node).

An application failure occurs when an application running on the platform fails in a manner *detectable* by the platform. Depending on the application and the service contract between the platform provider and the service provider, handling application failures could be the responsibility of the platform provider or the service provider (or both). In the former scenario, application semantics that constitute a failure will need to be specified a priori to the platform provider and the platform will need to incorporate application-specific mechanisms to detect and recover from such failures.

A capsule failure occurs when an application capsule fails in a way *undetectable* to the platform provider, for example an internal deadlock condition in an application. Capsule failures must be assumed to be the responsibility of the service provider and the platform itself does not provide any support for dealing with them.

We have found this factorization of failure types highly useful in designing fault-tolerance mechanisms for a public computing platform.

III. Status of On-going Work

We are building a public computing platform that addresses the requirements outlined in the previous section. In this section, we describe some of our initial research on yield management, resource control mechanisms for application isolation in such platforms.

We have begun investigating techniques for yield management in a public computing platform. These techniques involve attributing notions of trustworthiness and criticality to individual applications and providers, and using these attributes to overbook resources [14]. The key challenge in designing these capsule placement and admission control techniques is that traditional metrics such as utilization and predictable performance guarantees are no longer adequate. In a public computing platform, these techniques will be driven by the need to maximize yield. Admission control and placement based on this new metric can yield results different from those using more traditional metrics, and consequently novel techniques are needed to address this issue. For instance, as explained earlier, a cost-benefit analysis of admitting each new application and the resulting impact on overbooking is necessary in this approach, in addition to the traditional focus on the ability to meet performance guarantees.

We are also investigating resource control techniques for a public computing platform. The two canonical techniques for single-node resource control developed by the multimedia research community are *reservations* [1], [16] and *shares* [17], [15]. Whereas a reservation-based approach allocates resources in absolute terms (e.g., 2ms of CPU time every 20ms on a node), a proportional-share approach enables relative allocation of resources. In the latter approach, each capsule is assigned a weight and receives resources in proportion to its weight (allocation is relative because the share of each capsule depends not only on its weight but also the cumulative weights of the remaining capsules). In a pure-reservation-based approach, each capsule always receives *at most* its requested fraction; any unused bandwidth is wasted. In the proportional-share approach, a continuously runnable application always receives *at least* its assigned share and possibly more if other capsules do not utilize their allocations (i.e., unused bandwidth is redistributed among runnable capsules in proportion to their weights). Conceptually, resources requirements specified using reservations are upper bounds, while those specified using weights are lower bounds. Rather than wasting unused bandwidth, it is possible to modify a reservation-based approach to redistribute unused bandwidth among competing applications. Similarly, it is possible to combine proportional-share scheduling algorithms with admission control to limit the number of applications in the system and provide guarantees on delay and throughput [18]. Due to these similarities, it has been shown that reservations and shares are duals of one another [19] in the sense that a single scheduler can simultaneously allocate resources based on weights and reservations.

We are currently investigating resource control mechanisms that employ a novel combination of these two approaches. Our approach employs a reservation-based cluster-wide hierarchy; application providers can use this hierarchy to specify their aggregate requirements as well as those of individual capsules. Once an application is admitted and its capsules are mapped to individual nodes, the platform translates these reservations into equivalent shares and employs a proportional-share scheduler to enforce these allocations. Since the number of capsules at each node is constrained by admission control each application can be provided with guarantees on processor bandwidth and latency. This approach is conceptually equivalent to using a reservation-based scheduler at each node that can reassign idle bandwidth. Moreover, the hybrid approach permits a judicious combination of work conserving behavior and predictable allocation.

IV. Conclusions

We believe that there are compelling reasons to host large numbers of Internet services on a cluster-based platform. In particular, we are interested in the case where there are many more services than machines – this is a different space from current commercial hosting solutions, but one where we feel considerable innovation in applications is possible if the economic barrier to entry is very low.

Facilitating this innovation requires support for highly diverse resource guarantees: current application-level connection scheduling work restricts applications to web-based or similar request-response systems, and consequently restricts the diversity of feasible services (and how cheap it is to offer them). Much research from the field of multimedia systems can be reapplied here – indeed this may be a more compelling case for resource control facilities in the real world than multimedia workstations.

However, both the clustered environment and the business relationships involved in the design of public platforms adds new challenges: (i) heterogeneity of applications, distributed application components, and processing nodes; (ii) place of capsules within the platform; (iii) failure handling in a domain of split responsibility, and (iv) overbooking and yield management. Our current research focuses on these issues for a public computing platform.

ACKNOWLEDGMENTS

The authors would like to acknowledge the suggestions of Bryan Lyles in writing this paper.

REFERENCES

[1] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, and R. Fairbairns, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE JSAC*, vol. 14, no. 7, pp. 1280–1297, 1996.

[2] O. Spatscheck and L. L. Peterson, "Defending Against Denial of Service Attacks in Scout," in *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation*, February 1999.

[3] V Sundaram, A. Chandra, P. Goyal, P. Shenoy, J Sahni, and H Vin, "Application Performance in the QLinux Multimedia Operating System," in *Proceedings of the Eighth ACM Conference on Multimedia, Los Angeles, CA*, November 2000, pp. 127–136.

[4] J. Nieh and M. S. Lam, "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications," in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, Saint-Malo, France*, October 1997.

[5] 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*, New Orleans, Louisiana, March 1999, pp. 45–68.

[6] A. Chandra, M. Adler, P. Goyal, and P. Shenoy, "Surplus Fair Scheduling: A Proportional-Share CPU Scheduling Algorithm for Symmetric Multiprocessors," in *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI 2000), San Diego, CA*, October 2000.

[7] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-Based Scalable Network Services," in *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, San Malo, France*, October 1997.

[8] M. Litzkow, M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988, pp. 104–111.

[9] S. D. Gribble, M. Welsh, E. A. Brewer, and D. Culler, "The Multispace: an Evolutionary Platform for Infrastructural Services," in *Proceedings of the 1999 Usenix Annual Technical Conference*, Monterey, California, June 1999.

[10] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A mechanism for Resource Management in Cluster-based Network Servers," in *Proceedings of the ACM Sigmetrics 2000, Santa Clara, CA*, June 2000.

[11] Neil Stratford and Richard Mortier, "An economic approach to adaptive resource management," in *Proc. 7th IEEE Workshop on Hot Topics in Operating Systems (HotOS VII)*, March 1999.

[12] Barry C. Smith, John F. Leimkuhler, and Ross M. Darrow, "Yield management at American Airlines," *Interfaces*, vol. 22, no. 1, pp. 8–31, January-February 1992.

[13] T. Roscoe and B. Lyles, "Distributed Computing without DPEs: Design Considerations for Public Computing Platforms," in *Proceedings of the 9th ACM SIGOPS European Workshop, Kolding, Denmark*, September 17-20 2000.

[14] T. Roscoe, B. Lyles, and R. Isaacs, "The case for supporting risk assessment in systems," Sprint Labs Technical Report, May 2001.

[15] P. Goyal, X. Guo, and H.M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," in *Proceedings of the First USENIX Symposium on Operating System Design and Implementation (OSDI'96), Seattle*, October 1996, pp. 107–122.

[16] M. B. Jones, D. Rosu, and M. Rosu, "CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities," in *Proceedings of the sixteenth ACM symposium on Operating Systems Principles (SOSP'97), Saint-Malo, France*, December 1997, pp. 198–211.

[17] K. Duda and D. Cheriton, "Borrowed Virtual Time (BVT) Scheduling: Supporting Lantency-sensitive Threads in a General-Purpose Scheduler," in *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles (SOSP'99), Kiawah Island Resort, SC*, December 1999, pp. 261–276.

[18] P. Goyal, S. S. Lam, and H. M. Vin, "Determining End-to-End Delay Bounds In Heterogeneous Networks," *ACM/Springer-Verlag Multimedia Systems Journal*, vol. 5, no. 3, pp. 157–163, May 1997.

[19] I. Stoica, H. Abdel-Wahab, and K. Jeffay, "On the duality between resource reservation and proportional share resource allocation," in *Proceedings of the ACM/SPIE Conference on Multimedia Computing and Networking (MMCN'97), San Jose, CA*, February 1997, pp. 207–214.