New Resource Control Issues in Shared Clusters

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 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 provides computational resources to a large number of small *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 machines. 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 examples is workgroup clusters: a cluster of compute servers shared by a workgroup or 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 resource allocation 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. [**?**], schedulers, and abstractions (e.g. [3]).

Many of the advances above 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. However, our research has shown that existing resource allocation techniques do not directly generalize to multi-resource environments (multiprocessors, clusters), necessitating the design of novel techniques for such environments [4].

B. Cluster-based computing platforms

Much work has been performed recently on the use of clustered computing platforms for network services (see [5] 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 [6]. Similarly, intra-cluster security is relaxed as a simplifying assumption within the platform[7].

One notable exception to this is recent work on providing differential service to web-based applications, for example Cluster Reserves[8]. 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. Most, though not all, services on the Internet today fall into this category. In the future, however, 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.

However, techniques developed for existing cluster-based platforms for managing the cluster can still be highly appropriate for managing a public computing platform, providing that the additional requirements for security and resource control are met.

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. Requirements of a public computing platform

A. Resource Control Mechanisms for Heterogeneous Applications

In the recent past, server clusters have been employed for specialized scenarios such as dedicated hosting where each application runs on a dedicated node, and for providing replicated services where the application is replicated on all nodes of the cluster (e.g., cluster-based web servers). A public computing platform subsumes these two extremes by allowing applications to run on an arbitrary subset of the nodes. Moreover, since applications share resources on the platform, these subsets can overlap in arbitrary ways. We refer to that component of an application that runs on a given node as a *capsule*; each application can have one or more capsules, but not more than one per node.

Applications running on the platform are assumed

to be inherently heterogeneous. We envisage a mix of applications such as streaming audio and video serviers, game servers (e.g., Quake), vanilla web servers, and ecommerce applications. Observe that 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. In addition to heterogeneity across applications, there could be heterogeneity *within* each application. For instance, an ecommerce application might consist of capsules to service HTTP requests, to handle electronic payments and to manage product catalogs. Each such capsule might impose a different performance requirement.

The above examples illustrate the need to handle heterogeneity both across and within distributed applications. 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. The platform should be able to determine whether sufficient resources exist to meet these needs and reserve these resources on appropriate nodes. Further, it should employ resource control mechanisms to enforce these allocations on a sufficiently fine time-scale. As argued earlier, these issues are well understood for single node environments but these techniques do not carry over to multi-resource (multi-node) environments. For instance, it was shown in [4] that uniprocessor proportional-share scheduling algorithms can cause starvation or unbounded unfairness when employed for multiprocessors. Consequently, novel resource control techniques need to be developed to meet the performance requirements of distributed applications in public computing platforms.

B. 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. Such an algorithm should meet several requirements. First, placement of a capsules to nodes should be done incrementally without having to recompute the placement of existing capsules; further this mapping should be "optimal" or "near-optimal" to maximize revenue. Second, since a platform provider may add nodes to the platform or nodes may fail, it should be able to reconfigure the placement of modules dynamically. Such reconfigurations are also necessary when an application provider requests additional (or fewer) resources for the application based on variations in its popularity. All such reconfigurations should minimize the number of capsules that need to be moved from one node to another (since they involve stopping and restarting of capsules, a potentially disruptive operation).

Note that capsule placement is more than a simple bin packing problem, where capsules are assigned to nodes such that no node is saturated. Applications in a public computing platform can be mutually antagonistic. More seriously, they could be untrusted and could deny service to other applications. Hence, it is critical to isolate antagonistic or untrustworthy applications from one another. Performance isolation via resource control is necessary but not sufficient to address the issue of trust. A capsule placement algorithm should take into account trust (or lack thereof) among applications while mapping capsules to nodes. Another issue that impacts capsule placement is criticality. 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 the application. 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. Consequently, capsule placement becomes a multi-dimensional optimization problem—one that takes into account the trustworthiness of an application, its criticality and its performance requirements [**?**].

C. Handling Failures

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. A platform failure can also occur due to resource exhaustion: 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 will result 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). A special case of a platform failure is a platform-wide failure—a catastrophic failure that occurs due to multiple simultaneous node failures or the failure of a critical cluster-wide component (e.g., the cluster-wide resource manager). We assume that a typical platform-wide failure will require human intervention.

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 into the platform.

D. Tracking Resource Usage

A public computing platform should also employ efficient techniques to track resource usage so as to facilitate billing, long-term capacity planning and offline diagnostics. Due to the large number of nodes and the even larger number of applications in the platform, the sheer volume of usage-based statistics and the overheads of monitoring and collecting these statistics can be overwhelming. Consequently, novel resource monitoring and logging techniques need to be developed that can capture relevant information in a succinct manner and also correlate and aggregate information from different nodes in the cluster.

III. Status of On-going Work

We are designing a public computing platform that addresses the requirements outlined in the previous section. Our initial research focus has been on the design of resource control mechanisms and security mechanisms for application isolation in such platforms.

We started by examining two canonical techniques – reservations [1], [10] and shares [11], [12] – for allocating resources to applications in a cluster. 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 [13]. Due to these similarities, it has been shown that reservations and shares are duals of one another [14] 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.

We are also developing novel security mechanisms to isolate untrusting applications from one another. Our approach consists of dynamically programming the cluster interconnect using packet filters to isolate applications from one another.

Over the next few months, we plan to focus on (i) the design of cluster placement algorithms that take into account various factors such as trust, criticality and performance, and (ii) mechanisms to handle failures in a graceful manner.

IV. Conclusions

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 Journal on Selected Areas in Communications*, 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 3rd USENIX Symposium on Operating Systems Design and Implementation*, February 1999.

[3] Gaurav Banga, Peter Druschel, and Jeffrey 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.

[4] 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.

[5] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier, "Cluster-Based Scalable Network Services," in *Proceedings of the 16 ACM Symposium on Operating Systems Principles, San Malo, France*, October 1997.

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

[7] Steven D. Gribble, Matt Welsh, Eric A. Brewer, and David Culler, "The Multispace: an Evolutionary Platform for Infrastructural Services," in *Proceedings of the 1999 Usenix Annual Technical Conference*, Monterey, California, June 1999.

[8] Mohit Aron, Peter Druschel, and Willy Zwaenepoel, "Cluster Reserves: A mechanism for Resource Management in Cluster-based Network Servers," in *Proceedings of the ACM Sigmetrics 2000 International Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA*, June 2000.

[9] D. Sullivan, R. Haas, and M. Seltzer, "Tickets and currencies revisted: Extensions to multi-resource lottery scheduling," in *Proceedings of the 1999 Workshop on Hot Topics in Operating Systems (HotOS VII),Rio Rico, AZ*, March 1999.

[10] 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.

[11] K. Duda and D. Cheriton, "Borrowed virtual time (bvt) scheduling: Supporting lantency-sensitive threads in a general-purpose scheduler," in *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'99), Kiawah Island Resort, SC*, December 1999, pp. 261–276.

[12] P. Goyal, X. Guo, and H.M. Vin, "A hierarchical cpu scheduler for multimedia operating systems," in *Proceedings of Operating System Design and Implementation (OSDI'96), Seattle*, October 1996, pp. 107–122.

[13] 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.

[14] 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.