Resource Management for Internet Services Using Large-scale Clusters

Josep M. Blanquer, Antoni Batchelli, Klaus Schauser and Rich Wolski
Department of Computer Science
University of California Santa Barbara
{blanquer,tbatchel,schauser,rich}@cs.ucsb.edu

Abstract

The fast-paced growth of the Internet's user population is driving clustered architectures as the platform of choice for hosting high volume-services. The ability to provide a guaranteed service quality in such clustered architectures is becoming a need for today's e-businesses. Most companies that currently offer such service guarantees rely on over-provisioning their resources or on physically reserving groups of cluster nodes for different entities. Unfortunately, these attempts to address the QoS problem suffer from poor resource utilization, high cost and low flexibility. In this paper, we propose a non-intrusive, shaping technique that allows for the management of cluster resources such that service quality can be guaranteed to the clients. To this effect, we present Qflow, an attractive alternative solution suitable for the rapidly changing and ill-behaved traffic patterns of most Internet services. Qflow is more effective, has lower cost and is more flexible than any of the existing techniques. Simulation results of real scenarios are also presented, which validate and quantify the effectiveness of the proposed technique.

1 Introduction

Web-based services delivered using scalable cluster architectures are playing an increasingly important role in today's society. Key challenges facing these services include the exponential growth of users, unpredictable load fluctuations, and an increasing criticality to every day life. As users demand a wider array of new powerful services, scalable service capacity coupled with service quality guarantees become essential. Single-server solutions [7, 4, 5, 11, 1] are not adequate for current high-volume demands and the few existing clustered solutions require important operating system modifications [3] or tailor-made applications [13, 12]. So far, these challenges have been addressed through the application of parallel cluster architectures with simple resource allocation mechanisms including hardware partitioning and over-provisioning. In this paper we address the question of how to provide quality of service guarantees in software without relying on traditional hardware over-provisioning. Using a new technique for request-based traffic our results demonstrate how it is possible to achieve the same quality of service levels that hardware partitioning provides, but with improved resource efficiency.

In a commercial setting, client-side Quality of Service (QoS) at the service level (as opposed to the network level [6, 8] alone) provides inherent value by directly improving customer experience and strengthening customer trust in the offered services. In addition, Service Level Agreements (SLAs) that enable partnering between service providers also depend, critically, on service guarantees.

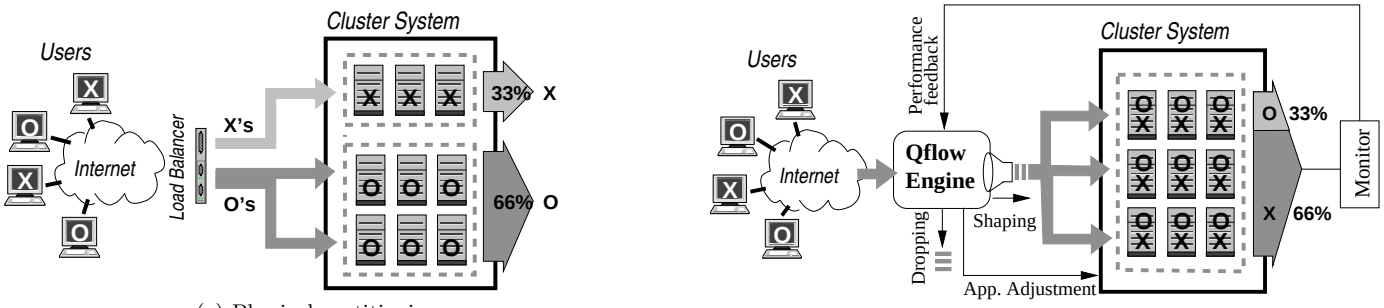
Preferential service prioritization is also emerging as a critical capability in many e-commerce settings. Clients often wish to pay for a specified level of service and the degree to which it can be guaranteed depends on individually changing client needs. Server provisioning mechanisms, such as load balancing, and portioning can provide limited prioritization capabilities, and typically do not permit prioritization levels to change dynamically in response to client demand.

At the same time, service providers must ensure that the computational resources they use to support their services are used efficiently. Extra resources that are kept on-hand to meet sudden, unpredictable "bursts" in request traffic are typically wasted during slack periods. The economic overhead (e.g. administrative and maintenance costs) associated with such over-provisioning must be balanced against the cost in lost business that a server over-run causes. Since bursts are difficult to predict, service providers must have the ability to reallocate resources to services dynamically as a way of mitigating load fluctuations while maintaining maximal resource efficiency.

Server resources themselves are often organized into computational clusters as a way of achieving low-cost parallelism and redundancy in an extensible framework with relatively scalable system administration properties. As the per-unit cost for cluster nodes continues to drop, and the quality of cheap cluster interconnect increases, the trend toward clusters and Internet service platforms will certainly continue.

2 Qflow: a New Approach

To provide both *client-side* QoS at the service level, and *server side* efficiency guarantees, we need a service infrastructure that supports flexible, high-performance, and dynamic resource allocation and control. In this paper we describe Qflow — a service-level control methodology that uses dynamic request shaping and admission control at the cluster boundary to provide efficient service guarantees. By controlling the shape of the request streams serviced by a cluster, and incorporating performance feedback in the shaping process, Qflow is able to achieve service quality guarantees with much greater resource efficiency than can be had



(a) Physical partitioning.

(b) Our approach: Qflow.

Figure 1: While traditional approaches provide service isolation by physically partitioning the cluster nodes, our approach consists of logically separating them to achieve a more effective resource utilization, lower cost and increased flexibility.

through resource partitioning, over-provisioning, or a combination of both.

The Qflow architecture is depicted in Figure 1.b. Cluster resources implementing various Internet services are guarded by a *Qflow Engine* that intercepts incoming client requests and controls their admission to the system. The Engine is parameterized by a control policy that dictates how and when (e.g., at what rate) requests for different types of services, depicted as Xs and Os in the figure, are forwarded on to the cluster. By controlling traffic at the cluster boundary, the system requires no knowledge or control over how the requests will be internally processed or how the services are internally organized. Similarly, the Qflow engine does not depend on the homogeneity of the nodes or the architecture of the cluster and it transparently complements any internal mechanisms such as load-balancing, service replication, internal node partitioning or node failure and recovery.

In addition to request shaping, the Engine is responsible for implementing controlled service degradation and prioritization. If the client request rate bursts beyond available capacity, the Engine intelligently drops requests so that clients paying for lower-priority service receive proportionally degraded service, and the resulting service degradation is predictable and stable.

The Engine requires a continuous stream of performance feedback data from the cluster of services it protects. A monitoring module intercepts the served requests at the cluster output and feeds service statistics back to the Engine. Based on such measurements and the current status of the scheduling algorithms, the QoS parameters of the Qflow Engine can be accordingly readjusted so that client-side QoS guarantees can successfully be achieved. In addition, if the internal cluster management infrastructure supports a control interface, the Engine may also feed control information forward, in response to increased client load, to affect a dynamic reallocation of resources. The deployment of the proposed architecture will enable cluster systems to provide flexible service guarantees without the need to modify or configure any hardware of software in the already working system.

Our experimental results show that using Qflow it is possible to provide the same level of service isolation as physical partitioning while maintaining a much higher resource efficiency. To do so, we will first describe the experimental methodology we have used to investigate the effectiveness of Qflow in Section 3 and present our findings in Section 4.

3 Experiment Methodology

To perform our experiments we have developed a customized simulator called *Mimic*. Mimic is designed to model the high-level behavior of cluster systems. Component based, it supports flexible and extensible node configuration with several types of load-balancer modules and server applications. It also emulates the basic O.S. resource management mechanisms such as a time-shared CPU scheduling along with network and disk resources. For efficiency, Mimic uses high level request objects rather than trying to emulate low level network packets and their protocol processing. In the experiments we have conducted for this investigation, we use a common server paradigm as in the apache web server [2] where a main process accepts all the incoming requests and forwards them to a preconfigured number of maximum helper processes.

3.1 Compared Techniques

At each of our experiments, we will compare the performance of Qflow with both *physical partitioning* and uncontrolled access (which we name *load-balancer*), under the same exact input traces and capacity-equivalent clusters.

Physical partitioning corresponds to the cases where a provider sells a fix portion of its cluster capacity to different companies. Such assignments are negotiated without any dependence to the input traffic characteristics. In this experiments, each entity would have a fixed number of dedicated nodes, corresponding to the desired proportion of the cluster.

The *load-balancer* case consists of a system with a level-7 load-balancer that will forward the incoming requests to the internal cluster nodes in a round-robin manner. Although this technique lacks of any QoS mechanisms, it will serve as a good comparison basis for its very high resource utilization.

3.2 Cluster Performance Regimes

To demonstrate the effectiveness of Qflow under a broad range of cases, we divide the space of possible cluster performance response into four categories shown in Table 1. For easier reference, we name the cases from A through D. The order in which the cases are named is done such that it corresponds to an increasing amount of total incoming traffic.

	underload	*overload*
imbalance	Case B	Case C
balance	Case A	Case D

Table 1: Cluster operation regimes.

The *imbalance* row corresponds to the cases where the service-request stream contains significantly more requests for one entity than for another. Conversely, the *balance* row represents the cases in which the request streams for each entity are either *all* under or *all* over their assigned minimum output guarantees. When the input traffic for all entities is below their guarantee (case A), the cluster is in underload and the output proportions should match the input ones (there's enough capacity in each of the static partitions). When one or more of the entities receive traffic above their guarantees (case B), a system will only be able to successfully service it all if unused cycles of the underloaded entities can be reused.

The column marked *underload* refers to the cases where the frequency and service load generated by overall request stream does not completely saturate all of the cluster resources. Finally, the *overload* cases are those where the overall request stream saturates all of the available resources, thus driving the cluster at its maximum output capacity. When the traffic exceeds the resources available for each entity (case D), the cluster is in overload and each partition should be functioning at it maximum rate. When on or more entities are above their minimum guarantees such that the total excess amount far surpasses the unused capacity from the other underloaded entities, we're in case C. The best a system can do in this case is to reuse all unused resources with the most valuable of the excess traffic.

3.3 The Case for a Weather Service

To demonstrate the efficiency and isolation capabilities of Qflow, we will use a common cluster-based scenario consisting on providing a single type of service (images) to several different partner portals. In particular, we will simulate a weather information site that serves precomputed weather forecast maps that can be linked from the weather sections of several partner electronic newspapers.

Each of the partners needs to be assured service protection against node or software misbehavior, and will be guaranteed a minimum share of the total cluster capacity that is directly proportional to the premium they are charged. The hypothetical QoS policy for the experiment consists on three different partners (CNN, NewYork Times and L.A. Times) which need to be guaranteed 70%,15% and 15% of the total cluster capacity, respectively.

The cluster size for the experiment will be of 20 nodes. Node partitions of 14, 3 and 3 for CNN, NYTimes and LATimes will be statically allocated in the case of physical partitioning. The maximum number of simultaneous requests for each apache-like server is set at 200.

3.3.1 Input Trace Characteristics

To present a realistic case and show how Qflow can gracefully handle bursty and ill-behaved traffic, we have used real traces from a web server for our experiments. The used traces are from Clarknet, a service provider that hosted web pages for companies and users [9]. The section used from the traces spans over a week worth of traffic and contains 1.6 million requests with one second granularity timestamps.

Additionally, we wish to vary the frequency with which requests can be presented to the system. To experiment with different possible requests frequencies, we artificially "speed-up" the Clarknet trace stream so that more requests are presented in each simulated time epoch. The advantage of this approach is that it preserves the arrival patterns present in the real-world trace. The disadvantage (from a comparison point of view) is that faster traces (ones with higher speed-ups) present greater segments of the trace to our simulator. We believe that any short-term correlation between request arrivals is an important characteristic to preserve. As such, we do not decimate "slower" traces so that they cover the same overall time period as the faster ones.

Finally, the execution (processor and network usage) times for each of the received requests have been derived from a heavy tailed distribution based on empirical profiling done with an apache process serving static HTML pages.

4 Experimental Results

In order to select the appropriate parameters for each of the tests at each of the four possible regimes, we first need to define and determine the available total capacity of the cluster.

4.1 Cluster Capacity

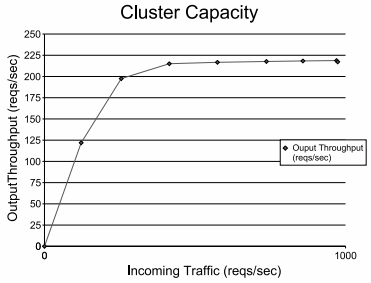


Figure 2: Replaying the input traces at different speedups using the 'load-balancer' case determines the cluster capacity for our experiment at 218 req/sec.

We define the maximum cluster performance as the total maximum throughput that the cluster is able to achieve when its resource utilization reaches 99% under a round-robin load-balancer configuration. To such capacity we use the *load-balancer* configuration and run several experiments using the same input traces at different speedups (Figure 2). The *load-balancer* case is designed to maintain all nodes busy given that there's enough input load. We identified the overload point at 218 reqs/sec where the bottleneck resource utilization (network) reaches 99.05%. To ensure that the presented results in this section fall within the *underload* or *overload* cases we have used input loads (trace speedups) that are far below or far above the cluster capacity breakpoint.

4.2 Resource Utilization

The first of the experiments we perform is designed to show the efficiency of Qflow in terms of resource utilization. Figure 3 plots the levels of resource utilization achieved for Qflow, physical partitioning and load-balancer at each of the 4 possible regimes. As expected, Qflow can successfully fulfill all the incoming requests in the cases where the cluster is below capacity (cases A and B). Physical partitioning is not able to serve all incoming requests in the unbalanced case B, given that one of the partition is fully saturated while the other have idle cycles. In the cases where the cluster is in overload (experiments C and D), Qflow can reach the maximum cluster capacity. Physical partitioning, again, cannot reach full capacity unless all partitions are always maintained fully occupied, regardless of a single partition being under extreme overload. The burstiness the used traces aggravates this problem, which causes physical partitioning to waste some valuable cycles (case D) when an entity goes below its capacity for very short periods of time even though in the average the input level is above its reserved partition. Qflow handles such common situations in a much more graceful manner given its immediate adaptation to highly dynamic changes.

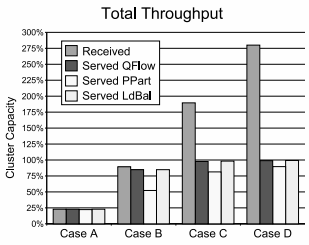


Figure 3: Res. Utilization

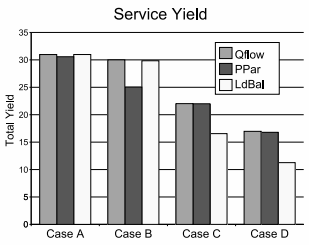


Figure 4: Service yield

The results presented confirm our expectations of Qflow achieving similar resource utilization in the balanced cases while being significantly better in the unbalanced cases.

4.3 Service Isolation

The other significant advantage of Qflow is its ability to provide the same level of isolation as physical partitioning despite having different entities sharing the same nodes with a time-shared operating system. Figure 5 shows the service proportions delivered by the cluster under the same set of cases. The shown proportions are normalized to the maximum cluster capacity. Input traffic is also normalized to the input rate at which the cluster reaches maximum capacity (i.e., the sufficient input traffic needed to drive the cluster to its capacity).

In the cases where the cluster does not reach its capacity (A and B), the output proportions should be equivalent to the input traffic given that all the incoming requests can be served without being dropped. As expected, Qflow always exhibits such behavior. However, in the case of physical partitioning, such output throughput is always limited by the size of each individual partition. This causes LATimes partition to reach only 18% out of the incoming 50% of the traffic in case B. It is worth noting that the minimum guarantees as shown in the graphs, can only be honored if there is enough input traffic demand to generate that much service. For example, in cases A and B, all incoming traffic for

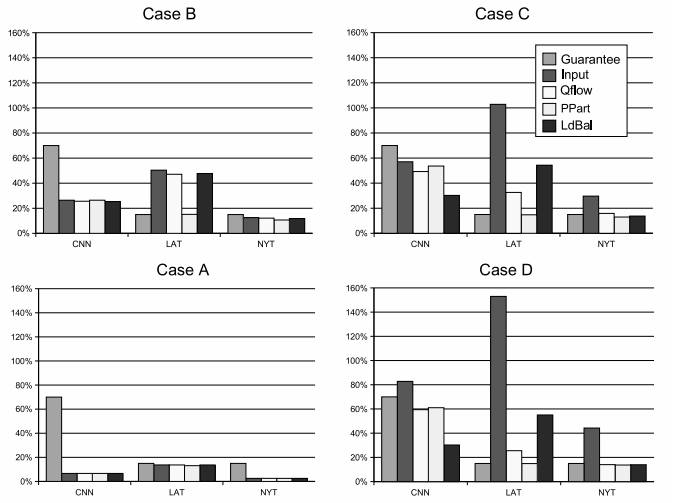


Figure 5: Service isolation results.

CNN was serviced although its minimum guarantee would have allowed to serve more, if more traffic had been received.

The benefits of service isolation become apparent when the cluster is running on overload (cases C and D). In these cases, the output levels should correspond to the defined proportions for each entity instead to the levels of received traffic. Both Qflow and physical partitioning are capable of honoring such guarantees unlike 'load-balancer' which still exhibits its proportionality to the input traffic. Furthermore, as it can be seen in case C, Qflow is again able to steal some unused cycles from CNN to serve the much overloaded LATimes clients. In case D, the guarantees for the CNN partner are not 100% fulfilled using either technique despite there is 'apparently' enough incoming traffic. The reason for this has to do again with the burstiness of the traces. That is, since the input traffic is not much higher than its guarantee, a sudden peak of traffic can cause the system to drop requests momentarily only to realize a second later that the entity is now idle and those could have been served if held a little longer. The queuing mechanisms of Qflow and the application dropping in physical partitioning exhibit a similar behavior in this situation. However, Qflow is able to at least get a better overall performance again since it can give these (unfortunately waisted CNN cycles) to the other entities (mostly LATimes in this case).

In short, the graphs conclude that Qflow is not only able to deliver the minimum throughput guarantees but it also shows that they are often surpassed in the cases where there's room for resource reassignment.

4.4 Service Yield

There are metrics other than service isolation by which we can measure the quality of the service a cluster is providing. For example, not all of the requests serviced by a cluster have the same importance. The "value" of a completed request may depend on both the entity it belongs to and the time in which was completed. Yield functions have been introduced as a way to express such value relationships [10]. Such relationships are commonly defined based on revenue values or premiums payed. They are unitless and their values are only

significant for comparison or analytical purposes. Examples of yield definitions range from constant functions (e.g., a completed request gets a value of 10 regardless of how long it took), to value-decaying functions for each different entity.

In the next section we will show that without deploying any intelligent dropping policy in the Qflow engine other than throwing away requests due to space constraints, we can still guarantee comparable or better service yield than physical partitioning. We expect the advantages to be much more significant when appropriate dropping policies are also implemented.

To calculate the total yield delivered by the cluster we need not only to account for the value of each serviced request but we also need to have into consideration the amount of requests that have arrived but not been served (lost business). We can factor in such unserved requests by including drops at 0 yield value:

$$Yield_{total} = \sum_{i=1}^{N} Yield(req_i)/(N + drops)$$

In our experiments, the value of a request completed is defined by a constant yield function and it is dependent on the entity it belongs to. The chosen constant values are 70, 15 and 15 for the CNN, NYTimes and LATimes such that it coincides with their proportion values.

4.4.1 Service Yield Results

As Figure 4 shows, for this experiment, Qflow always gets better service yield than physical partitioning in all the tested regimes.

In the cases where the cluster is underloaded (cases A and B), all the incoming requests can potentially be serviced without being dropped. This is the case for Qflow and load-balancer which achieve the same overall rate given that isolation is not a factor. However, physical partitioning needs to drop some requests given that some partitions can be momentarily full due to a sudden burst (Case A) or simply too much load (Case B). These have a detrimental effect on the yield achieved by the physical partitioning technique.

In the overloaded cases, service isolation becomes more important, given that servicing requests of more value will result in a higher yield value. In these cases both Qflow and physical partitioning clearly outperform load-balancer since they give preference to higher-value requests (while still honoring the guaranteed throughputs for each of the entities).

5 Conclusions

In this paper we have presented Qflow as an attractive alternative solution to the current techniques of physical partitioning and over-provisioning and showed that is suitable for the rapidly changing and ill-behaved traffic patterns of Internet services. We have also shown that for cluster systems Qflow is able to provide the same levels of service isolation as physical partitioning while achieving a much better resource efficiency.

The table below summarizes the observed performance of Qflow versus Physical partitioning for each of the described regimes. Note that this is precisely what one would expect since Qflow inherits both the QoS benefits of physical partitioning as well as the high efficiency of a round-robin load balancer.

	underload	*overload*
imbalance	Case B: **Qflow>PPart**	Case C: **Qflow>PPart**
balance	Case A: **Qflow≈PPart**	Case D: **Qflow≈PPart**

Another of the advantages of Qflow is its ability to be deployed transparently in any of the existing cluster architectures giving such systems the ability to isolate misbehaviors and to control how the service is delivered in a flexible manner. We believe such offered compatibility and transparency to be of great value given the existing investment in current cluster systems that have no service quality mechanisms. Qflow can also be used to protect from DoS attacks and unexpected overload situations due to 'too' successful marketing campaigns or 'news-of-the-day' effects.

Finally we have also shown that Qflow achieves a better service yield than physical partitioning. In the future, we plan to study smarter dropping techniques that take request service times and more sophisticated yield functions into consideration. The capability of including request servicing times in addition to service isolation guarantees will give Qflow more expressivity power, allowing more comprehensive policies to be enforced.

References

[1] J. Almeida, M. Dabu, A. Manikutty, and P. Cao. Providing differentiated quality-of-service in Web hosting services. In *Proceedings of the First Workshop on Internet Server Performance*, June 1998.

[2] Apache http server. http://httpd.apache.org.

[3] 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 Conference on Measurement and Modeling of Computer Systems*, Santa Clara, California, June 2000.

[4] G. Banga, P. Druschel, and J. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 1999.

[5] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, September 1999.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC 2475, Dec. 1998.

[7] J. Blanquer, J. Bruno, E. Gabber, M. Mcshea, B. Özden, and A. Silberschatz. Resource Management for QoS in Eclipse/BSD. In *Proceedings of the First FreeBSD Conference*, Berkeley, California, Oct. 1999.

[8] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: An overview. RFC 1633, July 1994.

[9] Clarknet traces. http://ita.ee.lbl.gov/html/contrib/clarknet-http.html.

[10] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated Resource Management for Cluster-based Internet Services. In *Proc. of the 5th USENIX Symposium on Operating Systems Design and Implementation*, Boston, MA, Dec. 2002.

[11] T. Voigt, R. Tewari, D. Freimuth, and A. Mehra. Kernel mechanisms for service differentiation in overloaded Web servers. In *Proceedings of the 2001 USENIX Annual Technical Conference*, Boston, Massachusetts, June 2001.

[12] M. Welsh, D. Culler, and E. Brewer. SEDA: An Architecture for Well-Conditioned, Scalable Internet Services. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, Banff, Canada, Oct 2001.

[13] H. Zhu, H. Tang, and T. Yang. Demand-driven service differentiation in cluster-based network servers. In *Proceedings of the IEEE INFOCOM*, Anchorage, Alaska, April 2001.