Cooperative Load Balancing and Admission Control in Distributed Systems

Ashish Kundu

IBM India Research Lab
Block-1, IIT, Hauz Khas
New Delhi - 110 016, India
E-mail: kashish@in.ibm.com

Abstract

Requests in a distributed computing environment lead to multiple processings, both concurrent and sequential at various components during the lifetime of that request. Any failure of a critical transaction would lead to failure of the complete process, therefore leading to dropping of that request. Such request drops have an adverse effect on the resource utilization of the system and the QoS guaranteed to the user. This paper studies the factor of request drop rate in a distributed system and its effect on the resource utilization and QoS. As part of the solution, an admission control framework using flow control technique has been proposed in order to minimize request drops. Multiple load balancers implementing the admission control techniques cooperate with each other in order to minimize the effect of request drops. A multiple-queue based architecture for a load balancer has been proposed for this purpose.

Motivation and Introduction

Analogy of admission control (and load balancing) in distributed systems with admission control (and routing) in network links is non-trivial. While comparing load balancers with routers and system components with networks, the following is worthwhile to look into: system components often comprise of statefull services/applications, whereas network links are mostly state-less. Latency of request processing in a component depends on many factors other than the size of request and load and capacity of the component, such as underlying request processing complexity etc., which vary from component to component and request to request. Latency of packet/message transmission in a network link depends on factors based on the size of the message and load and capacity of the link etc, yet a relatively more deterministic set of factors. While a single request processed by a system component may give rise to more than one request for next level component(s), in networks the number of messages does not increase. Definition of load is different from component to component, whereas in networks, load mostly refers to the same definition. The component-level complexity, local logic for processing of various kinds of requests, possibly large number of types of requests for each local component, statefull-ness, local definitions of load and capacity etc., make it extremely difficult to predict and/or benchmark load on a system component and its relation to qualities of service, thereby making admission control more locally bound in contrast to its global nature in networks. Given the local nature of load balancers, admission control can be appropriately coupled with load balancers.

Admission control of requests in systems decides upon requests to be allowed for processing or dropped, at the same time aiming to minimize the request drops. It can also be used to enforce Quality of Service (QoS)/Service Level Agreements (SLA). Since load balancers are the access points to systems and are responsible for dispatching of requests, they are suitable to implement admission control mechanism/policy for a system. We would treat the term – "load balancer" in the paper to be synonymous to "admission controller".

Prior to the description of the problem, let us define "request drop". *A request is said to be dropped*, (i) if, prior to the processing – partial or full - of the request by the system or a system component, it is eliminated from any future consideration for processing or (ii) if any transaction that is critical for the complete processing of the request is aborted or (iii) if another request, which is generated out of the processing of the original request and whose complete processing is critical for the complete processing of the original request is dropped. In other words, a request is said to be dropped, when the client of request is notified about the fact that *the request could not successfully processed/completed*. Use of term "request" throughout the entire context of this paper would be synonymous to terms such as "task", "job" and "query". Let us discuss various problems associated with request drops.

As request rate or task arrival rate in a system increases beyond a limit, the rate of requests getting dropped also increases. This increase in rate of requests dropped reduces system throughput, increases system response time, thereby worsening overall system performance, which in turn affects user satisfaction adversely. As quality of service (QoS) is affected, it becomes difficult to enforce service level agreements (SLA), if any. Transaction aborts resulted from requests drops at intermediate components in distributed systems may give rise to inconsistencies in system states; therefore such request drops may lead to system rollback and recovery. System rollback generally leads to large degradation in performance and effective resource utilization.

Request dropping rate acts as a possible indicative parameter for performance of associated load balancer, if any. Increase in request drop rate may indicate a drop in performance in load balancers. If some or all requests are being dropped at load balancers, response time and throughput of the load balancer worsen.

In order to enforce service level agreements (SLA), it is necessary to provide quality of service (QoS) differentiated on the basis of customers. Load balancers can facilitate differentiated QoS by providing differentiated distribution of requests. Load balancer would be using a factor to classify requests to various classes; it may be based on the QoS level and/or customer and/or other parameters associated with the request.

Let us look into each of these problems in detail. As request rate increases beyond a limit, requests get dropped at the load balancer and/or at the system. Requests would be dropped at load balancer due to the following reasons.

- Incoming request rate is higher than the maximum rate at which load balancer can receive requests.
- Buffer of load balancer is full. Sometimes load balancer buffer becomes full because,
    - Dispatching rate is less than the rate at which buffer is being filled up.
    - Possible target system (or system component) replicas are highly loaded; so load balancer is unable to dispatch requests. This in fact reduces the dispatching rate of the balancer, thereby acting as a reason for less dispatching rate.

In a centralized system, requests would be dropped due to the following reasons – either (i) the rate of incoming requests is higher than the maximum rate at which the system can accept; or (ii) the system cannot accept some requests due to some reasons such as request does not contain enough data so that it can be processed; or (iii) the system replica is overloaded, at the time of receiving a request and therefore, it drops the request. (Regarding

why load balancer would dispatch a request to a replica that is too highly loaded to handle the request. consider the example in Figure 8. It has a load balancer for *n* replicas of the centralized system S. Load balancer dispatches a request R to Si that is highly loaded. R is dispatched to the highly loaded Si, because, either the balancer does not use load in determining the target replica or it does not have the correct load information for Si and/or other replicas or its load balancing technique does not do proper balancing act. While the first and last case depend on the load balancing technique used, the second case depends on load monitoring technique, correctness of load data and frequency of load information updation at load balancer. We will not focus on these issues in this work.)

In a distributed system, requests would be dropped at the front-end component or at a component other than the front-end (we call them intermediate components). Requests would be dropped at a component because either (i) the component cannot accept the requests – the rate of incoming requests is higher than what the maximum rate at which the component can accept; or (ii) the system cannot accept some requests due to some reasons such as a request does not contain enough data so that it can be processed; or (iii) the component is too loaded to handle the requests. Each request served by a component at level *k-1* may give rise to more than one requests meant for a component at level *k*. So even if the original incoming request is possible to be handled by the system components, other requests might be dropped. This makes processing of original or sibling requests meaningless.(Regarding why load balancer would dispatch a request to a replica that is too highly loaded to handle the request, consider the example in figure-2. It has a load balancer (load distributor) that distributes incoming requests among two replicas of a distributed system (DS1 and DS2). A request is received by load distributor. Assume that the load distributor has employed load balancing technique that uses load information of the replicas in order to find target replica for a request. Load balancer might be getting load information either of the front-end or of the whole system in not-so-comprehensible[1] manner. Thus it may be oblivious of the relatively exact state of the intermediate components. Even if the load balancer receives load data for each component, the state of intermediate components would most probably have changed significantly by the time requests reach them after being processed at front-end component. In the example, LD uses load information of front-end and dispatches a request to DS2,. By the time, the request or requests generated out of processing of the original request reach App1 in DS2, App1 component has moved onto a highly loaded state. The request/requests then gets dropped at App1 after being processed by front-end of DS2.)

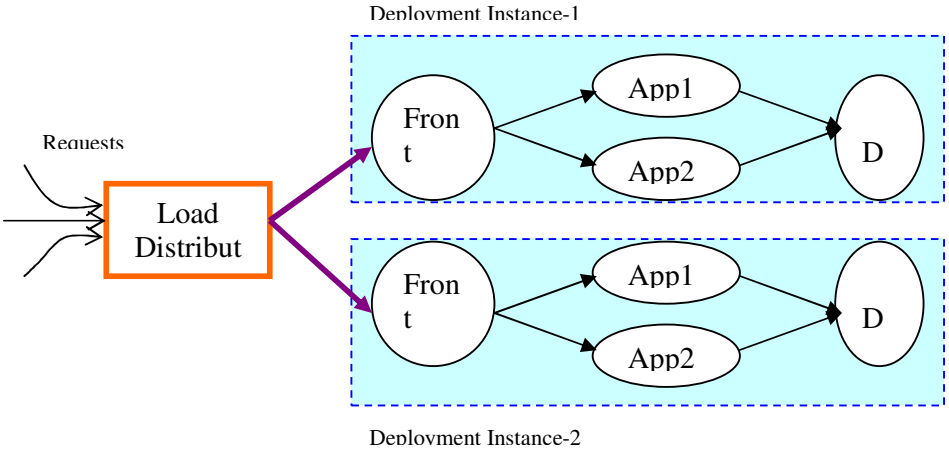
Resources used (wasted) to dispatch and/or process a request (and its child requests/transactions, if any) before it is being dropped could have been utilized for possible better quality of service for other requests that would be successful. The more is degree of such resource wastage, the more the system performance worsens. As rate of request drop increases at a load balancer, throughput reduces; response time of a load balancer increases (*i.e.*, worsens). It might also be possible to successfully serve some requests R that have been dropped, if resources utilized for some other dropped requests could instead be used for these requests.

An end-user of a request that is dropped would be more satisfied when failure of her request is notified to her quickly than when failure is notified after a delay. So the problem is that – can the end-user of requests be notified of failure of their requests as early as possible.

[1] For example, load is expressed as a single number. It may be generated (and normalized in the process) out of a set of various metrics of monitored data. But, the request drop rate is dependant on the function that generates this concise-but-not-comprehensive data. It is possible that many functions will arrive at a large number even though none of the components have large load value, whereas same function will arrive at a smaller value even though only one of its components have very large load value and others have normal load. This makes load balancers oblivious of the state of components.

Quality of Service (Service Level Agreements) based load balancing facilitates enforcement of QoS/SLA guaranteed to customers of the system. Therefore request buffering and dispatching, even differentiated request dropping has to be based on QoS/SLA.

Therefore, this work addresses the issue of request drop and proposes a admission control-



Legend: App1: Application 1 (Resource Class 1)
App2: Application 2 (Resource Class 2)
DB: Database system

cum-load balancing framework and an architecture for the load balancer/admission controller for the following objectives.

1. Reduction in the number of requests dropped at intermediate components.
2. Enforcement of QoS/SLA at load balancer
3. Reduction in the number of requests dropped at load balancers
4. Increase in throughput of load balancers
5. Differentiated load balancing

Objectives 2-4 pertain to both centralized and distributed systems, while 1 pertains to only distributed systems.

Figure 1: Traditional Load Balancing among replicas of a system

Related Work

Figure 1 is a typical load distribution framework in distributed systems as in literature. There is a single load distributor, distributing load among replicas of the *whole system*. Our pending patent [23] on virtual server farms uses load balancing at component level, i.e., per cluster in distributed systems. Figure 2 shows typical load balancing in centralized systems. In essence, an LD distributes load for the front-end of the system. Requests forwarded to the front end replicas may or may not be served completely before getting dropped at intermediate component(s). In literature, various load balancing techniques have been studied [24]. A number of patents [1,2,3,4,5,6] exist on load balancing techniques. All of them use the same load-balancing model. All of these patents suggest algorithms/systems for finding a target replica (out of a set of replicas) and then forward the request to that replica.

Earlier works [9,13,19] on flow-control based load balancing or cooperative load balancing defines cooperation amongst hosts/replicas, not among load balancers. Flow control based load balancing in [9] focuses on flow control of messages between processors

(target replicas) that carry out computations; flow control is employed by processors in a single cluster to share/distribute load among themselves (flow control among intra-cluster processors). [7], in its future work, talks about the need to design/develop cooperative load balancing system with the following intention – a set of load balancers have been assigned to carry out load distribution among replicas in a cluster. The load balancers in this particular set cooperate among themselves in detecting the target replica for a job. This is different from what is being addressed by the work – the load balancers cooperate with each other in order to better system performance, not to detect the target replica for the job. As per this work, cooperative load balancers, as defined, need to control dispatching of requests to their assigned clusters, based on the feedback from its directly attached (neighbor) load balancers. This enables flow control between clusters (inter-cluster). Some work exist for QoS aware routing at packet level [11]. The embodied work addresses load balancing based on application level QoS/SLA.

The following issues are not handled by the existing load balancing frameworks and load balancers.

- Requests are dropped at load balancers
- Requests are dropped at intermediate components in distributed systems.
- No mechanism to facilitate QoS/SLA enforcement.
- Lower throughput of load balancers than what is possible.

The effects of above deficiencies have the following implication on system performance.

- Poor resource utilization.
- Low throughput of requests.
- Higher average response time for requests.
- Dissatisfaction of end-users due to low throughput and high response time.

Even though to achieve high scalability, load balancing in a system would have to overcome these deficiencies, above deficiencies from an application–level load balancing point of view have yet not been looked into,.

Terminology

Logical Instance: A logical instance is either a physical instance/component or some capacity of a physical instance/component. A logical instance might be represented by a pair - (name/address of physical instance, capacity allocated in the physical instance). All use of term “instance” and “replica” in this paper is synonymous to “logical instance”.

Cluster: A cluster is a collection of logical instances that are identical in terms of functionality supported, protocol for using the instance etc.

Next-level load distributor: A next level load distributor of LD-x is an LD that is adjacent and topologically next to LD-x. For example, in Figure 2, LD-2 and LD-3 are next-level LD’s of LD-1; LD-4 is next-level LD of LD-2 and LD-3.

Previous-level load distributor: A previous-level load distributor of LD-x is an LD that is adjacent and topologically just previous to LD-x. For example, in Figure 2, LD-1is previous-level for LD-2 and LD-3; LD-2 and LD-3 are previous-level LDs of LD-4.

Front-end: In this paper, we define a front-end to be a cluster of application instances that receive requests from end users.

Request Classification Factor (RCF): The factor that is used to classify a request (differentiate from other requests) into a category. The work uses customer identity or service class/grade associated with the request or a combination of customer id and service class of a request as RCF to classify incoming requests.

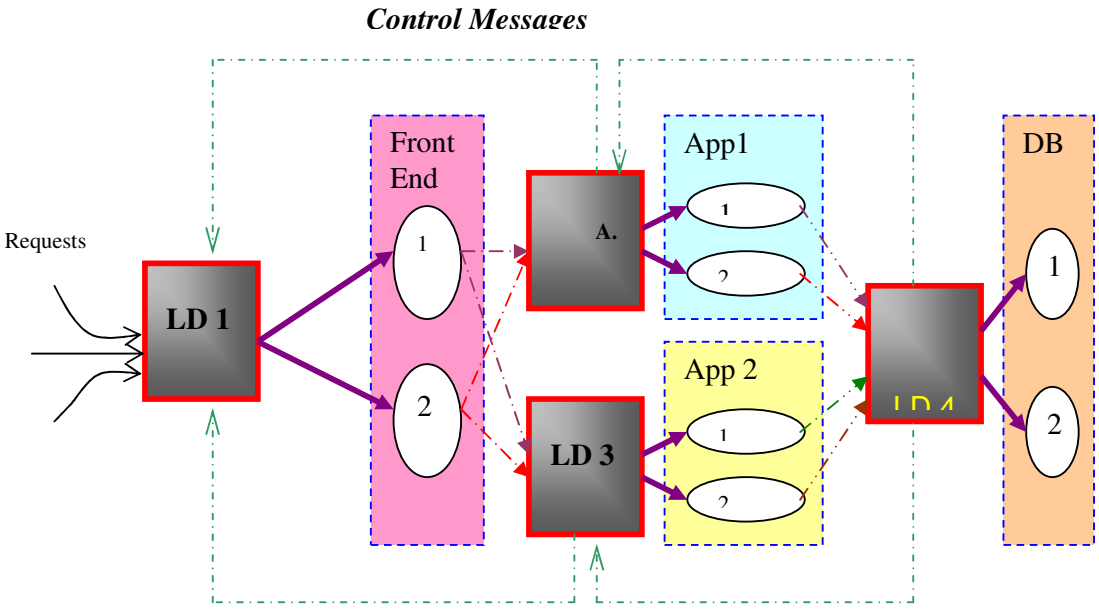
In the paper, onwards, we will use the terms request, task and job synonymously. Similarly CLD, LD, CLB, load distributor, load balancer will be used synonymously.

Solutions to the Deficiencies in Existing Systems
The load balancing framework uses the following measures in order to solve the deficiencies in earlier work (problems mentioned earlier in problem definition section).

1. *Cluster level load balancing/admission control*: In order to reduce request drop at system components, a load distributor per cluster of the distributed system is used (as shown in Figure 2 and 4). In either of the proposed load balancing frameworks shown in Figure 2 and 4, there is a load balancer per cluster of instances of the same application. It also reduces request drop at load balancer, as the dispatching rate of load balancer might increase.

2. *Flow control among load distributors/admission controllers across clusters*: In order to reduce requests dropped at intermediate components, flow control of requests (cooperation mechanism) among inter-cluster LDs: An LD cooperates with its previous-level and next-level LDs and changes its operation and request-dispatching rate dynamically. An LD receives control messages from its next-level LDs and it sends control messages to reduce or increase its request dispatching rates. This mechanism reduces the probability of requests getting dropped at the intermediate level. Figure 2 is the proposed framework that uses this technique. Figure 2 does not use this mechanism and drops requests as per the following illustration. Assume that, request rate for LD-1 at one point is 100 at one point of time. It has got a capacity to handle request rate more than 100. LD-1 determines the target instance in cluster-1 for each request R1 and forwards R1 to that instance. Processing of the requests results in a number of requests – say 250 requests. 150 of them is meant for cluster-2 and 100 meant for cluster-3. LD-2 has a capacity of handling 110 requests per unit time. LD-2 therefore would drop 40 requests. LD-3 has a capacity of handling 150 requests per unit time. In the mean time LD-3 receives 100 requests from cluster-1. Though the number of requests is within its capacity, it may not be able to dispatch all the 100 requests. This is because, the instances in cluster-3 collectively can serve only 75 requests at this point of time, as some of the previous requests have generated very high load. Thus 25 of the requests have to be dropped. Upon processing, 110 requests in cluster-2 generated 180 requests for cluster-4. LD-4 has a capacity of 150. So at least 30 of them have to be dropped. These 110 requests in cluster-4 may have generated high load. So LD-2 has to drop any new incoming requests. Most of the requests generated in cluster-3 meant for cluster-4 will be dropped at LD-4, since cluster-4 might have used up all its capacity to serve 110 requests from cluster-2. In the above scenario, requests are being dropped after being processed in earlier clusters. Such requests may generate large load on these earlier clusters (such as cluster-1 and cluster-2), thus preventing some new requests from being served. Requests are dropped due to load distributor is not able to handle so many requests or due to high load on the instances or both. Such a solution increases resource utilization, better user response time; increases request throughput and better user satisfaction.

3. *Dual Buffering of requests by Load Distributor*: In order to reduce request drop at load balancer and in order to do QoS/SLA based load balancing, Cooperative Load Distributors (CLD) use two-level request buffering mechanism (also called as dual-queuing mechanism)**.** First buffer is for all incoming requests. Each second level buffer contains requests that have a common factor among themselves. The factor is based on customer and/or service class of the request. First level buffer can handle the (near-)



Legend: LD 1: Load Distributor – 1
App1: Application 1 (Resource Class 1)
App2: Application 2 (Resource Class 2)
DB: Database system

Figure 2: Cooperative Load Balancing Mechanism for system in Figure-1

overflow state of one or more of second level buffers, thereby reducing requests dropped.

4. *Flow control of requests among two level buffers in an LD*: In order to reduce request drop at load balancer and in order to do QoS/SLA based request dropping if at all necessary, flow control of requests among two levels buffers in an LD is employed.

5. *Parallel Dispatching of requests*: In order to increase throughput of load balancers and reduce their response time, each CLD is capable of dispatching requests from each of the second-level queue concurrently thus giving parallel dispatching capability. This is intended to increase the throughput of the load balancer as well as that of the system as a whole.

"LD per cluster framework with flow control" cuts down the number of requests that would have been dropped in case of "LD per cluster framework with no flow control". In the former case, feedback messages allow load balancer(s) with free buffer capacity at some previous

level(s), if any, to forward requests at a controlled rate, so that buffer overflow at next level distributors can be averted.

Differentiated request dispatching contributes towards enforcement of Quality of Service (QoS)/SLA policies. Cooperative Load Distributors can dynamically add or remove buffers for new or existing classes of requests, respectively. CLDs facilitate increase and decrease in the size of buffers dynamically.

Overview of the Framework

A centralized system employing cooperative load balancing framework involves a set of replicas of the system (Figure 1). These replicas are to be load balanced for incoming requests. A distributed system employing cooperating load balancing framework involves clusters of its components. Replicas of the whole system (as in Figure 1) are not used. Cluster of replicas of a component is used. As in figure-3, each cluster (say cluster-2) is for one component (such as App1) and it consists of replicas, only of that component (App1). A load balancer distributes load among these replicas in a cluster. So we have moved from the typical framework as in figure-2 to the framework shown in Figure 2. In order to reduce request drops at intermediate clusters, flow control is employed among load balancers across clusters (figure-4). Each load balancer that senses about possible request drop at future sends out flow control messages to its previous-level load balancers. This flow-control message is used to direct the previous load balancers to reduce their request dispatching rate of requests (of a specific request class, if any). Most of the tiered architectures and cluster-level architectures of distributed systems do not involve loops (directed acyclic graph). So this feature moves the framework as shown in figure-3 to the one shown in figure-4. Since existing load balancers lack in flow control mechanism, a new load balancing system has to be designed that can generate and implement flow control. In order to enforce QoS/SLA at the load balancer level, a load balancer should be capable of carrying out differentiated load balancing. Differentiated dispatching of requests leads to parallel dispatching of requests, thereby increasing load balancer throughput and reducing its response-time. We call such a load distributor a cooperative load distributor (CLD).

A CLD accounts for dynamic allocation and de-allocation of instance capacity to customer(s) by resource managers, if any in the system. It also uses dynamic load information collected by load monitors, if any in the system, to effectively balance load. CLD dynamically determines whether to send any control message to previous-level CLDs on the basis of the current incoming request rate per customer and request dispatching rate per customer and/or capacity allocation/de-allocation or based on any other system state.

How the System Works

The framework for load balancing facilitates load balancing on per cluster basis and flow control among inter-cluster load balancers. The system of cooperative load distributor carries out load balancing of incoming requests to target replicas on a cluster and performs dual buffering, differentiated load balancing of requests, parallel request dispatching, intra-load balancer flow control, distributed flow control among inter-cluster load balancers. The framework is shown in figure 2 and the system is shown in Figure 3.

The system of CLD has the following components – Global Request Handler (GRH), Request Dispatching Manager (RDM), Specific Request Handler (SRH) and Load Distribution

Controller (LDC). GRH includes a buffer and a request receiver, and a request classifier. Request Dispatching Manager receivers of messages from LDC, GRH and senders of messages to SRH, LDC and GRH. It also contains optionally, a load receiver and a message sender for load monitors. Specific Request Handler includes a request receiver, buffer called second level buffer (SLB) and a request dispatcher (RD). LDC includes receivers for messages from various components and adjacent load balancers and resource manager, senders of messages to such components, a module for initialization of the system when it comes up, a module to update the CLD records due to any dynamic modification in the cluster, a module that modifies the configuration of CLD dynamically based on a new request class, or based on any change in the cluster capacity etc, an optional module to implement any QoS/SLA related intelligence – *e.g.*, it intelligently decides the buffer sizes per QoS class based on their frequency of becoming full, rates of request drop etc, an optional module to do intelligent load balancing and/or flow control.

GRH is the first level of dual-buffering used and a set of SRHs comprise of the second level buffers. The number of SRHs is identical to the number of request classes. GRH is primarily responsible to receive incoming requests, optionally carry out admission control based on authentication/any other criteria, accept the request, and then classify each request and send the request to an SRH appropriate to the class of the request. Each SRH is specific to a request class. Each SRH is primarily responsible to receive requests from GRH and buffer them in its buffer, if it is not full or else drop the request, dispatch each request based on the load balancing technique used. GRH and RDM have to carry out flow control in order to implement flow control among GRH and all SRH. RDM is responsible to carry out management of SRH and dispatching rate of each of the SRH. LDC is responsible to manage the whole LDC system and apply flow control among load balancers across clusters.

As and when GRH receives a request, it checks, if its buffer is full or not. If it is full, it drops the request or else accepts the request. After accepting, GRH buffers the request in GRB – global request buffer. RC (request classifier) classifies requests in the buffer and dispatches them to their proper SRH. In order to classify application-level requests (if the load balancer is doing application level load balancing), RC parses a request header, determines the request classification factor. If the load balancer is doing network-level load balancing, then RC parses the IP packet header for its priority and/or other information (dependent on what kind of RCF is used – implementation details) and determines the factor. After determining the factor, RC hands over the request to the appropriate SRH.

If SLB of SRH is full, then SRH drops any request that it receives (Flow control reduces these types of requests); else SRH buffers the request in SLB. RD (request dispatcher) of each SRH dispatches requests to the target replicas based on the load balancing technique used. RD needs to know how to dispatch the request to a target replica. Since request dispatching techniques used is different in network level and application level load balancing, RD is also a separately implemented for each of them. For application load balancing, request dispatching varies from one application to another depending on what protocol each of them use. So for a class of applications, there will be one implementation of RD and for another class, another one implementation. If the load balancing technique is static one, no load information is (effectively) necessary; e.g. round robin load balancing techniques. If it is a dynamic load balancing technique, then RD might require the load information on each replica on the cluster for the specific request class (to which SRH belongs). This load information is received from the load monitor(s). The dispatching rate of RD is maximum when the load balancer starts up. Since RC in GRH and RD in SRH are dependent on

whether the CLD is being used for application level load balancing or network-level load balancing, they can be implemented as pluggable components for the load balancer. Such a pluggable component would be used by the load balancer during its start-up. This component is application-dependent for application-level load balancing.

RDM keeps track of the state of each of SLB. As and when an SLB becomes near-full (for example, 75% full) RDM sends a flow control message to GRH. (near-full state is always less than or equal to the full state of a buffer. Near-full state is useful as, by the time GRH enforces flow control, GRH might have sent out some more requests for the specific request class.) The flow control message includes the request class and the incoming request rate for that request class among other things. GRH might stop forwarding requests or reduce the rate at which it is dispatching requests or can take some other measures. As and when an SLB state flips from "near-full" state to "not-near-full" state, RDM sends out a message to GRH about the state change. GRH then may go back to its earlier state. As and when GRB becomes near-full or GRH observes that some SLB is becoming near-full more frequently or GRH is dropping requests as GRB is full or incoming request rate is much higher than the rate at which it dispatches them to SRHs or similar situation arises, GRH sends a control message to LDC about the state. This message contains, relevant data – incoming request rate, request classifying rate, request dispatching rate of SRHs etc. LDC then decides whether to increase the buffer sizes of GRB/SLBs and/or to send a flow control message to its previous-level load balancers.

LDC is responsible to decide, when to generate and send flow control messages to its previous-level load distributors. When GRH is near-full, or at least one SLB is near-full or an SLB is getting near-full quite frequently etc., LDC may send a flow control message.

LDC, upon receiving a flow control message for a request class r, decides how much to reduce the dispatching rate for r, how much time to stop dispatching requests for r, how much increase in buffer size to be made (in order to handle requests that will be buffered due to these two previous decisions). If it does not know the latest request dispatching rate for r, then it may ask RDM through a control message for that value. It will compute the values and its decisions and then direct RDM to implement the final request dispatching rate, the time for which request dispatching to be stopped and/or increase in SLB size. Similarly, if there is any directive regarding this for GRH, LDC directs it also. RRC in RDM then controls the dispatching rate of the SRH according to the directive. RDM/GRH increases the buffer size of SLB/GRB, respectively, if required. GRH and RDM sends out positive acknowledgements to LDC for successful implementations of the directive, or else negative acknowledgements are sent out to LDC. Upon successful implementation of a flow control directive, LDC sends out its new dispatching rate of request class r to all its next-level distributors. Such CLDs, upon receiving the message update their records.

As and when, there is significant increase in the capacity of SLB or capacity (in terms of request rate) of some instance(s) in the cluster for request class r, LDC will notify the previous level load balancers about the increase in capacity, projected (due to increase in capacity) maximum request rate that it can handle, its current dispatching rate for r, its current incoming request for r etc. LDC will notify only if, it has recently sent a flow control message to its previous load balancers for request class r. As soon as an LDC receives such a message, it computes how much increase d is possible in its request dispatching rate for r. Then LDC directs GRH and/or RDM to increase its dispatching rate of r.

Due to some resource manager messages or some other reason (*e.g.*, to counter frequent overflow of an SLB or GRB), LDC might decide to change (increase/decrease) the size of any of the buffers. To change the size of GRB, LDC directs GRH. GRH implements the directive and sends a positive response upon successful change or else a negative response. To change the size of an SLB or to change the dispatching rate of an SRH, LDC directs RDM informing it about the action required for the specific request class. RDM implements the action in the directive and sends a positive response upon successful change or else a negative response.

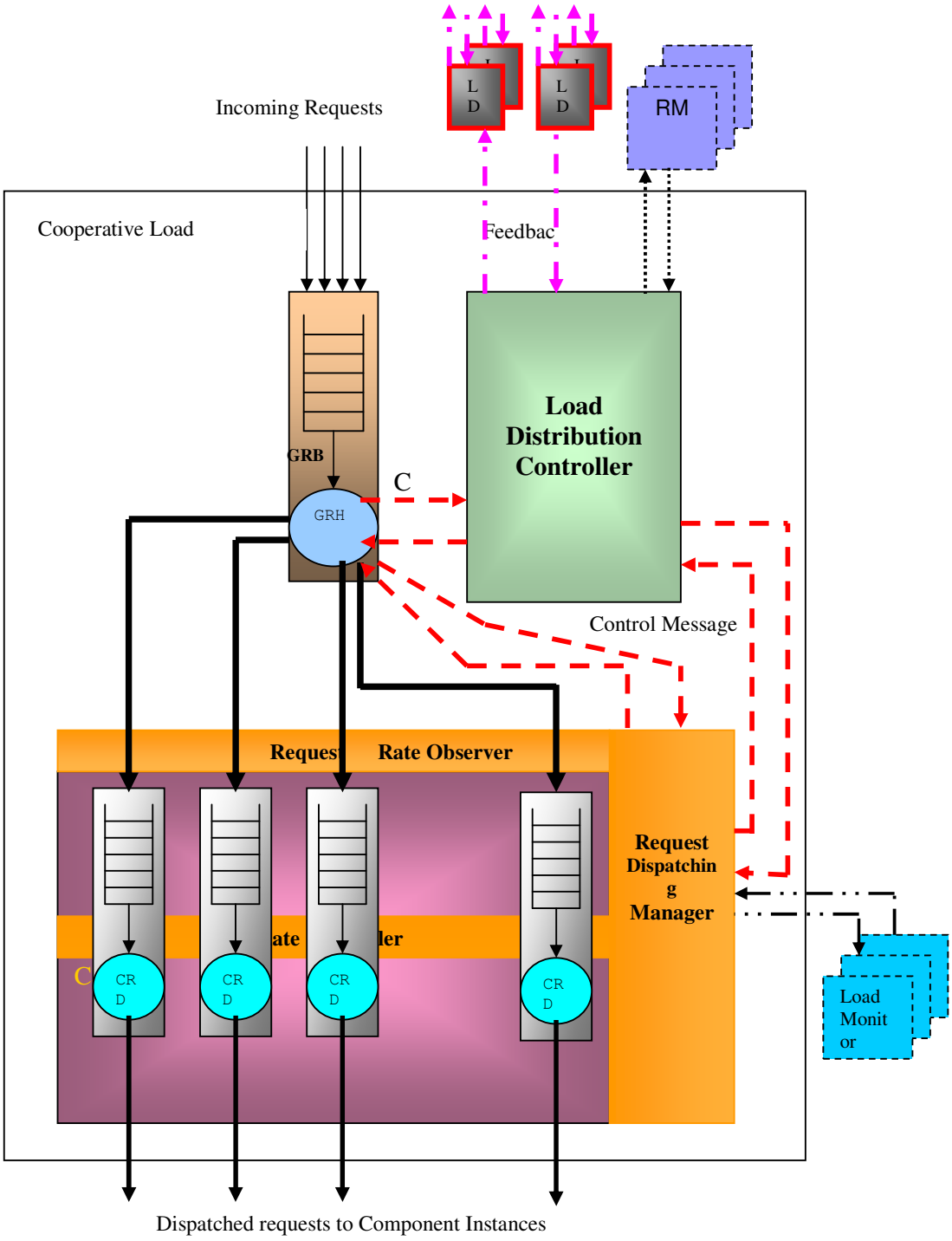
As and when, the resource manager adds a new request class r, it notifies LDC of a CLD regarding the update. The update will also contain the possible QoS/SLA parameters and/or possible maximum request rate[2] for r for that cluster. LDC decides upon the possible changes to GRB size and possible size of SLB for r. LDC directs RDM to create a new SRH for r and add r to its list of request classes, so that RRC and RRO can enforce their functionalities for r as well. RDM creates an SRH with specific the specific SLB size as specified by LDC. RDM then, sends a message to LDC informing about the successful SRH creation. If RDM cannot create an SRH, it sends a negative response to LDC. Upon receiving a positive reply from RDM, LDC directs GRH for change in GRB size and addition of new request class r. GRH implements LDC directive by increasing GRB size and updating its list of request classes. GRH sends a positive acknowledgement upon successful completion. If GRH fails, then it sends a negative response. If any kind of fault at GRH/LDC/RDM did not allow the load balancer to effectively add request class r to load balancer, then LDC might send a negative response to resource manager, if necessary.

Similarly for removal of a request class r, LDC decides about the possible change in GRB size. It sends a directive to RDM to remove the corresponding GRH. RDM upon receiving such a directive sends a control message to GRH saying "removing SRH for request r". GRH then, stops receiving any further requests for r from outside, forwards all requests of r to the SRH. Then GRH sends an acknowledgement to RDM. Upon receiving the acknowledgement from GRH, RDM waits till SLB for r becomes empty (there is no more request to be dispatched) and then removes the SRH. Then it sends an acknowledgement to LDC. LDC then directs GRH to change the size of GRB and to remove request class r from its records, as appropriate. GRH changes the size of GRB, updates the list and then sends a positive response to LDC. Upon receiving positive responses from both GRH and RDM, LDC then might send an acknowledgement containing "request class r has been removed" to RM, if required. If any kind of fault at GRH/LDC/RDM did not allow the load balancer to effectively remove request class r, then LDC might send a negative response to resource manager, if necessary.

Requests of each request class are dispatched concurrently with requests of other request classes. While the CLD uses parallel dispatching, in order to ensure that no two dispatchers send requests to the same instance at a time, each logical instance should belong to only one resource class. It can be recalled that, a logical instance is either a physical instance/replica or some capacity of a physical instance/replica in a cluster. For example, if request classes are Gold and Silver, a logical instance would belong to either of Gold and Silver, but not both. In [26], each request class belongs to a customer and each logical instance belongs to only one customer.

[2] Request rate for class r, to front-end cluster is the aggregate request rate from all end-users of request class r at a given point. But each request of class r in a front-end cluster might generate more than one requests for next clusters. So for each cluster, there is possibly a different incoming request rate.

Sequential dispatching technique can also be employed for this architecture (shown in Figure 8). Instead of parallel dispatching, sequential dispatching can be implemented using following technique. Request Dispatcher of each SRH are removed. Instead of these request dispatchers, a single request dispatcher for all request classes is employed. This request dispatcher is responsible to carry out scheduling among the various queues based on any parameter (*e.g.*, QoS/SLA based weight) and dispatches one request at a time.



Legend: GRB: Global Request Buffer, GRH: Global Request Handler, CM: Control Message, CRD: Client Request Dispatcher, C_i: Request for $Client_i$, CRB: Client Request Buffer (associated with CRD), RM: Resource Manager.

Figure 3: Architecture of Cooperative Load Distributor

Detailed Description
Figure 3 is the block diagram of the cooperative load distributor. It consists of the following components –Global Request Handler (GRH), Request Dispatching Manager (RDM), Specific Request Handler (SRH) and Load Distribution Controller (LDC).

Global Request Handler (GRH): Functionalities of GRH are as follows.

- Receive and classify requests
  - Accept/drop requests
  - Buffer incoming requests
  - Classify requests based on RCF and
  - forward them to appropriate SRHs
- Request rates
  - Keep track of incoming overall request rate (and incoming request rate per request class) averaged over a time period
  - Keep track of rate of requests sent to per customer averaged over a time period.
- Flow Control
  - If GRB is near-full, send control message to LDC
  - When a SLB is near-full, send control message to LDC
- Implementation of LDC directives
  - Increase/decrease size of GRB
  - Stop/start forwarding of requests to SRH for a request class
- Response to RDM
  - Send acknowledgement to RDM regarding removal of SRH/reduction in size of SLB
- Response to queries
  - Send incoming request rate averaged over a period of time (overall/for a request class) to LDC and RDM, whenever requested.
- List of Request classes
  - Maintain and update a list of Request classes based on the values of Classification Factor (RCF)
  - Maintain and update a list of Specific Request Handlers (SRH) corresponding to request classes.

Main functionalities are to buffer incoming requests, determine request classification factor (RCF) of the request and hand over the request to appropriate specific request handler (second level) that handles that class of request. In case, there is no RCF for the cluster, then the job of GRH is to hand over a request to the only one SRH in the balancer.

GRH consists of Global Request Buffer (GRB) and Request Classifier (RC). GRB is used to buffer requests. RC classifies requests based on the defined RCF and then hands them over to the appropriate SRH.

GRH receives incoming requests to the cluster, which the CLD is associated with. GRH buffers incoming requests in a Global Request Buffer (GRB). Size of GRB is more than the Second level buffers (SLB) associated with RD. GRH can dynamically increase and decrease the size of GRB upon receiving directives from LDC. There is a minimum limit to the size of all buffers including GRB. Size of GRB may increase because of induction of new value(s) of RCF, new instance(s), allocation of more capacity or increase in capacity of existing instance(s) in the cluster. Size of GRB may decrease because of removal of existing RCF value(s), existing instance(s), de-allocation of capacity or decrease in capacity of existing instance(s) in the cluster. A GRH may not consider the new resource allocation(s) ready for

receiving requests unless and until, it receives a final message from LDC. Size of GRB also increases due to large overflow frequency of some SLB(s). GRH maintains a list of RCF values. GRH keeps track of incoming request rate per customer averaged over a time period. This is used by LDC as a factor for computing the feedback message to its previous-level CLDs, whenever required.

It also keeps track of rate of requests sent to SRH averaged over a time period per customer. This is used by LDC as a factor for computing the amount, by which the dispatching rate of requests of a specific SRH to be reduced, whenever necessary.

GRH receives message from Request Dispatching Manager as and when a SLB becomes near-full or a near-full SLB is not any more near-full. Upon receiving this message, GRH sends control message about "SLB_i is near-full" and incoming request rates averaged over a period of time for each such customer $\boldsymbol{i}$, to LDC. *Near-full* state of a buffer is defined by CLD implementation/configuration. It determines to what extent the buffer is full. For example, a buffer is near-full when 90% of its capacity is already occupied by requests. Definitions of *near-full* for GRB and SLB may differ. GRH keeps track of all SLBs that are near-full at a point of time. This list is used to perform intra-load distributor based flow control of requests. If GRB is free enough, then GRH holds requests in GRB and stops dispatching them to SLB. GRH holds requests having those RCF value(s), whose SLB(s) is (are) near-full.

RC picks up requests from GRB; identifies the RCF value associated with each of the requests; RC hands over the request to that SRH if the SLB is *not* near-full. Otherwise, GRH holds the request in its buffer until the SLB does not remain near-full any more. Buffering of requests, holding of requests of near-full SLBs, picking up of requests from GRB depend on the buffering policy implemented by GRH. Identifying the RCF value of a request is dependant on how RCF has been defined. Definition of RCF is generally based on QoS/SLA information in order to enforce QoS/SLA.

A near-full state of GRB leads to following actions from GRH. GRH informs LDC about the state of GRB and the incoming request rate averaged over a period of time. LDC decides which class of requests to be dropped and which class of requests to be buffered and lets GRH know these two sets. The decision depends on the SLA policy implemented by the load distributor. LDC implements this SLA policy. Which requests and how these requests are to be buffered, is dependant on the buffer management policy implemented. As soon as GRB consumption reduces below the near-full limit, GRH informs LDC about the state. The component also sends incoming request rate averaged over a period of time upon a request from LDC. GRH keeps track of incoming request rate averaged over a period of time ΔT. ΔT is defined by configuration. So the other previous-level clusters, which feed the cluster of CLD with requests, determine value of ΔT for the CLD.
Upon receiving a near-full message - "resizing SLB for request class *i*" - from RDM, GRH stops forwarding requests to that SRH until it receives a message saying SLB is not near-full. The message - "resizing SLB" - stops GRH from notifying LDC about overflow, if any, of SLB for that SRH. Upon receiving a "removing SRH" message from RDM, GRH checks if any more request of that customer is in GRB. If so, it sends all those requests, then sends an acknowledgement to RDM that allows it to remove the SRH.

Request Dispatching Manager (RDM): Functionalities of request dispatching manager are as follows.

- Request Rates

    - Keep track of average request drop rate per request class of over a time period
    - Keep track of average dispatching rate per request class of over a time period
- Modifications
    - Direct SRHs to increase/decrease size of their second level buffers (SLBs)
    - Direct SRHs to increase or decrease their dispatching rates of requests.
    - Inform each SRH about the set of logical instances and capacity allocated in each instance for the specific request class at the time of creation of SRH and after each change capacity or instance.
    - Inform GRH as and when SLB size to be reduced or SRH to be removed.
- LDC directives
    - Create and/or destroy SRH as per directive from LDC
    - Implement changes as required by LDC
- Flow control
    - When SLB_i is near-full, send control message about "SLB_i is near-full" to GRH
    - When a near-full SLB_i is not any more near-full, send control message about "SLB_i is not near-full" to GRH
- Response to Queries
    - Send request drop rate per request class, request dispatching rate per request class averaged over a period of time to LDC, whenever requested
    - Inform LDC about the dispatching rate of requests for request class(es) averaged over a time period, as and when required.
- Lists/tables
    - Maintain and update a list of logical instances for each request class
    - Maintain a list of request classes and corresponding mapping to SRHs.

RDM keeps track of the average rate at which requests get dropped per customer over a time period Δt as defined by implementation. This information is sent to LDC whenever demanded by LDC. RDM increases or decreases the size of second level buffer as directed by LDC. Size of an SLB may increase because of allocation of more capacity in the cluster to a request class. Size of an SLB may decrease because of de-allocation of capacity in the cluster from a request class. Size of an SLB also increases due to its high overflow frequency, high average request drop rate. RDM may not consider the increase in resource allocation(s) ready for receiving requests unless and until, it receives a final message from LDC. RDM creates new SRH for each new request class and destroys an existing SRH for a removed request class. RDM waits for an acknowledgement from GRH to remove an SRH or reduce the size of an SLB. This is to ensure that GRH does not forward any more requests to that SRH after RDM removes the RD.

When SLB_i is near-full for a set of request classes S, RDM sends control message about "SLB_i is near-full for *i,* a request class in S" to GRH. When a near-full SLB_i is not any more near-full, SRH sends a control message - "SLB_i is not near-full for *i,* a request class in S" to GRH. This message ensures that hereafter, class ***i*** is treated normally by both GRH and LDC.

RDM can increase or decrease dispatching rate of requests for request classes individually upon receiving directive from LDC. RDM implements a subcomponent Rate Controller to perform this task. LDC asks RDM to increase or decrease the request dispatching rate of request class(s) in order to enforce flow control of requests among clusters. Rate is increased for a request class, if earlier it was decreased and later, the class has been allocated more resource capacity in the next-level cluster(s). Similarly, rate is decreased, if the second level

buffer(s) has gone near-full in the next-level cluster(s) and this has triggered feedback control message(s) from next-level cluster(s). LDC also directs RDM to increase/decrease the dispatching rate of a class by some amount. RDM asks SRH to dispatch according to the new rate. RDM also keeps track of dispatching rate of requests per request class averaged over a time period.

To reduce the size of an SLB, RDM first checks if the new size is less than or equal to the near-full limit with respect to the current size. If yes, RDM sends a near-full message with extra information – "resizing SLB for request class ***i***" to GRH. Then RDM informs SRH to reduce the SLB size. This is to avoid dropping of requests at SRH level. Similarly before removing an SRH, RDM sends a message to GRH – "removing SRH for request class ***i***", waits for an acknowledgement from GRH; then waits for SLB to be empty; then RDM removes RD.

Specific Request Handler (SRH): Functionalities of a request dispatcher are as follows.

- Receive requests
  - Buffer requests forwarded by GRH.
- Load balancing –
  - Pick a request from buffer,
  - Determine target instance using load balancing technique implemented by RD
  - Dispatch the request to the target instance, if found at all.
- Request rates
  - Keep track of requests dispatched over a period of time.
- Implementation of directives
  - Implement new (increase or decrease in) dispatching rate of requests as directed by Request Rate Controller.
  - Implement new (increase or decrease in) size of second level buffer as directed by RDM.
- Lists
  - Maintain and update a list of logical instances in cluster for the request class.

SRH has two major sub-components – Second Level Buffer and Request Dispatcher (RD). Buffering of requests in second level buffer and picking up requests from it are based on the buffer management policy implemented by SRH. *Near-full* state of each buffer is also defined either separately for each SLB or identically with each other. RD implements load balancing technique to be used to determine the target instance for a request in buffer. Determination of a target instance can be based on the load information received. The target instance has some or full capacity allocated to the customer. If a target instance is determined, then **RD** dispatches the request to that instance else **RD** may keep the request in buffer. **SRH** responds to the actions taken by Rate Controller to increase or decrease dispatching rate. One possible method is – Rate Controller informs a new delay period between two dispatches of requests and **RD** implements that dynamically changed delay period. Delay of zero quantity increases dispatching rate to maximum possible rate and a positive delay period reduces the dispatching rate. Other methods can be used for this purpose. Similarly RDM can direct RD to increase or decrease the size of SLB by some amount.

Load distribution controller (LDC): Following are the functionalities of a load distribution controller.

- Control messages

    - Buffer (flow) control message(s) from next-level CLDs
    - Send (flow) control message(s) to previous-level CLDs
    - Receive control messages from GRH and RDM
- Initiation of flow control
    - Generate a feedback message and send it to the concerned previous-level CLDs
- Implementation of flow control
    - Decide the action(s) to take upon processing a feedback message from a CLD
    - Compute the amount by which dispatching rate of requests for request class(s) to be increased or decreased
    - Compute the amount by which GRB or SLB(s) size(s) to be increased/decreased
    - Send new dispatching rate(s) to next level load balancers.
- Implement resource manager allocations plans
    - Add/remove new request classes
    - Modify list of logical instances in the cluster
    - Modify QoS/SLA parameters associated with a request class
- Directives
    - Direct RDM to create/destroy a SRH
    - Direct GRH to modify size of GRB
- Response to queries/initiation of queries
    - Respond to queries from resource manager on current request drop rate, request dispatching rate…
    - Query RDM/GRH for request rates or states of SLB/GLB/SRH…
- Load distributor configuration
    - Initialize the cooperative load distributor during startup
    - Optionally allow dynamic changes to configurations

Upon receiving a feedback message, LDC of CLD-k interprets the message and then does the following. If the message contains a request dispatching rate for class r, of its host CLD - CLD-k as X, while the current request dispatching rate for class r is Y, where X is significantly more than Y, then LDC does not do anything; it discards the message or else it does the following. If the message contains a non-zero duration for which request dispatching for a request class r has to be stopped, LDC immediately stops dispatching of requests by RD for that request class. LDC asks RDM to stop the dispatching, RDM in turn asks RD to stop the dispatching of requests. LDC computes its new dispatching rate of requests for the request class. The technique used to compute such a value is implementation dependent. An example technique is given below.

Example: Flow control message M is from CLD-m to CLD-k. It contains, total incoming rate of requests for class r at CLD-m is R_r^m and expected incoming rate is E_r^m, its dispatching rate D_r^m. Let the dispatching rate for class r, at CLD-k be D_r^k.

Ratio of contribution to incoming rate $\mathbf{C = D_r^k / R_r^m}$.
Change to dispatching rate $\mathbf{z_r^k = C * (E_r^m - R_r^m)}$.
New dispatching rate $\mathbf{d_r^k = D_r^k + z_r^k}$.

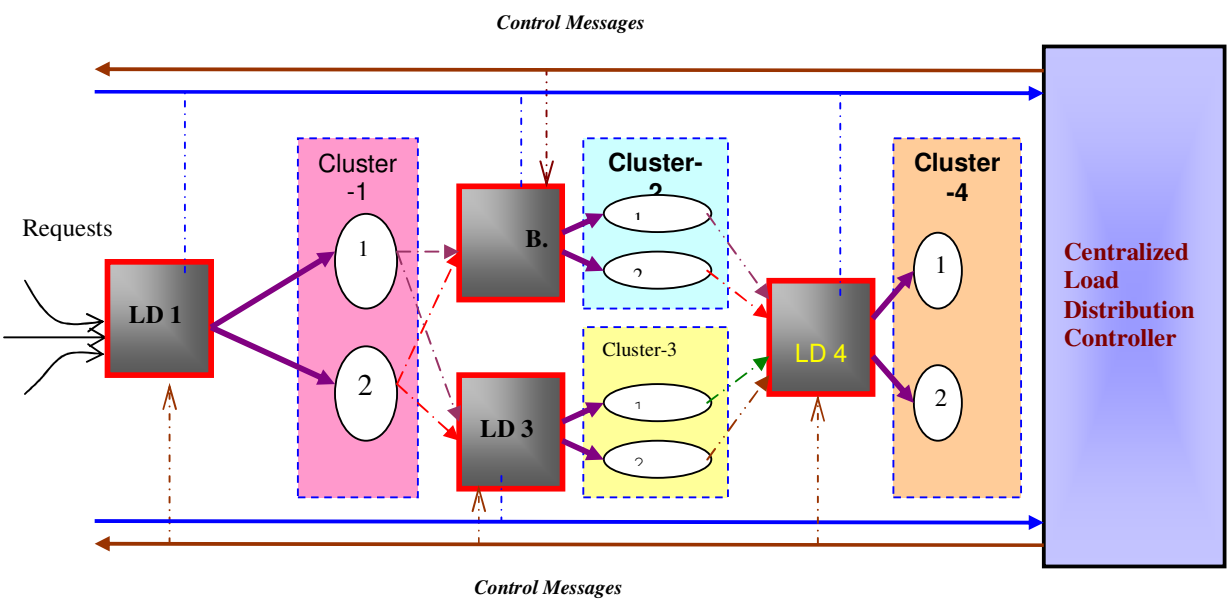
LDC then informs RDM about the new rate. RDM directs the specific SRH of the request class to implement the new dispatching rate. LDC receives a positive acknowledgement from RDM upon successful implementation of the directive (of course any directive). Then it sends out the new dispatching rate for the specific class to all its next-level load distributors.

If the feedback is about more resource allocation and an increased expected rate of requests for a request class, then LDC decides whether to increase dispatching rate in proportion to the expected rate and if so, how much. Decision of not increasing dispatching rate or computing the amount of increase, is influenced by the condition – whether other clusters in next-level having same request class will suffer from near-full states consequently.

LDC generates feedback messages in the following cases: If LDC receives message about GRB to be near-full or SLB to be near-full or more resource has been allocated to a request class, for whom a feedback message has earlier been issued to reduce flow of requests at previous-level CLDs.

Load distribution controller can be used to implement SLA policies if any in the system. SLA policy implemented is a factor in determining which request class to be given more buffer size and which request class less. SLA policy also determines, in case of GRB or multiple SLBs being near-full simultaneously, which request class(s) request dispatching rate not to be reduced beyond a level, which request class(s) request drop rate to remain the least possible. In case, GRB is near-full or multiple SLBs are full, LDC can send feedback message as well as add more buffer space at least for all those request classes, whose SLBs are near-full and whose request drop rates are to be minimized. Request dispatching rate for some request classes are not to go below a minimum amount; for such request classes, LDC decides to add more buffer space to their SLBs, if possible. LDC avoids sending feedback messages for such request classes. In case of more resource allocation to a request class, whose dispatching rate has been reduced by sending feedbacks to previous-level CLDs, LDC determines whether to send a feedback message to previous-level CLDs to increase the dispatching rate for this request class; computes what is the expected rate of incoming requests; sends this expected rate and current incoming rate to all its previous-level CLDs as part of the feedback message. If GRB is near-full, then LDC may decide to increase the size of GRB.

Message from resource manager can be for increase or decrease in resource allocated to a request class or induction or removal of a request class. To implement increases/decrease in resource allocated to a request class, LDC increases SLB size, sends a feedback message, if needed or decreases SLB size, if needed. SLA policy, if any implemented may determine each of LDC decisions. Induction of a request class is implemented by directing RDM to add a RD, (thus implicitly making sure the request class is also removed from RDM request class list) and to add the request class in GRH list of request classes. Size of SLB is as per the resource allocated by resource manager. LDC performs resource amount to buffer size mapping. Such functionality is present in every LDC. Upon increase in size of SLB, size of GRB is increased by non-negative amount. In order to remove a request class, LDC directs RDM to remove its RD, thus implicitly making sure the request class is also removed from RDM request class list. LDC directs GRH to remove such request classes. Request drop rate averaged over a time period per request class from request rate observer can be used by LDC in computing the feedback messages to be sent. LDC buffers incoming feedback messages in a buffer called LDCB (Load distribution controller buffer). LDC, implements a buffer management policy.



Legend: LD 1: Load Distributor – 1

Figure 4: Alternate Solution: Centralized Cooperative Load Balancing Mechanism

Upon startup of CLD, LDC gets the initial list of request classes, the instance allocation list per request class and other startup information through some configuration mechanism. CLD may have been implemented to use a specific RCF or to determine the definition of RCF during its startup. The configuration information may include a formally specified definition of request classification factor (RCF), that the GRH can interpret. LDC creates GRH, RDM and asks RDM to create SRHs and other subcomponents. It informs GRH and RDM about the request class list. Instance allocation list is sent to RDM; RDM later sends appropriate segments of this list to each of RDs.

Centralized flow control among inter-cluster load balancers: In the given solution, flow control is distributed and is carried out at LDC of each load balancer. As an alternate solution, there can be one or more centralized LDCs who carry out flow control in a centralized manner. All other functionalities of LDC such as

- Implement resource manager allocations plans
- Direct RDM to create/destroy a RD
- Receive control messages from GRH and RDM
- Initialize the cooperative load distributor during startup

can be carried out at an entity (a partial LDC) local to each load distributor. But this solution is not going to scale as good as the distributed solution as the centralized LDC can be the bottleneck and then single-point of failure. The corresponding load balancing framework is shown in Figure 4.

Multiple levels (>2) of queuing inside a load balancer: Dual level queuing i.e. queuing levels of two is necessary to differentiate among requests and facilitate implementation of SLAs as well as handle overflow requests. But multiple levels of queuing is a technique can also be implemented to get around of the given solution. The more number of the levels

inside a load balancer is, the more is the latency inside the load balancer. Moreover, the intended functionality that requires using multiple level queuing with levels > 2 can be implemented using dual level queuing. Also in order to have queuing of levels > 2 contains the dual level queuing and thus is not an alternate solution in its entirety without infringing upon the embodied work. For example, an alternate solution of load balancer can have first queue as a global buffer and second level as buffers on service class based. (*Note – by using such a hierarchy, the model uses our work and thus is not completely an alternate solution*) and third level buffers as per customer based. Such a model can be easily implemented as a dual level queuing model, where first level buffer is as usual - global buffer and second level buffers are per service class per customer based.

It should be noted that since, this problem or set of issues have been looked at for the first time, the alternate solutions stated can also be claimed as our work.

Conclusion and Future Works

This paper describes a new load balancing framework and architecture of a load distributor for distributed systems. A service provider/hosting environment can use this framework in order to provide better managed resources and better performance of the system. The new features of the framework are:

- Load distribution at the level of each cluster in a distributed system
- Flow-control among load distributors across clusters.

The load distributor architecture supports mechanisms for

- Receiving dynamic load information per replica in a cluster from load monitors, thus supporting use of dynamic load balancing algorithms.
- Receiving state of the system, new capacity plans for the cluster/replica from resource managers, thus supporting dynamic load balancing algorithms and dynamic resource management.
- Sending load balancing data such as current request drop rate for a service class, dynamically to resource managers.

An experimental analysis of request drop rates in various distributed systems and the flow control techniques would be part of the future work. Furthermore, various factors such as parallel dispatching and multi-level buffering have to be implemented to find out the resource utilization and QoS improvements.

References

1. US5983281: Load balancing in a multiple network environment
2. US6092178: System for responding to a resource request
3. US5938732: Load balancing and failover of network services
4. US5915095: Method and apparatus for balancing processing requests among a plurality of servers based on measurable characteristics off network node and common application
5. US5506999: Event driven blackboard processing system that provides dynamic load balancing and shared data between knowledge source processors
6. US5495426: Inband directed routing for load balancing and load distribution in a data communication network
7. The Design of an Adaptive CORBA Load Balancing Service, Ossama Othman, Carlos O'Ryan, and Douglas C. Schmidt, Distributed Systems Engineering Journal, April, 2001

8. Dynamic Load Balancing in Distributed Multimedia Systems, Atsunobu Hieaiwa, Naohitsa Komatsu, Kazumi Komiya, and Hiroaki Ikeda, 40th Midwest Symposium on Circuits and Systems, August 3-6 1997, California.
9. Flow Control and Dynamic Load Balancing in Time Warp, M. Choe and C. Tropper, Proceedings of the 33rd IEEE Annual Simulation Symposium, 16 - 22 April 2000 Washington, D.C.
10. Load Balancing in Distributed Workflow Management Systems, Li-jie Jin et al, ACM Applied Symposium on Applied Computing, 11-14, March 2001, Las Vegas, NV.
11. QoS-Aware Routing schemes Based on Hierarchical Load Balancing for Integrated Services Packet Networks, C. Casetti et al, IEEE International Communication Conference (ICC'99), Vancouver, Canada, June 6-10, 1999.
12. Improved Strategies for Dynamic Load Balancing, Chi-Chung Hui et al, *IEEE Concurrency*, volume 7, number 3, pp. 58-67, September 1999.
13. A Cooperative Load Balancing Game in Distributed Systems, Daniel Grosu et al, 4th Workshop on Advances in Parallel and Distributed Computational Models, April 15, 2002, Fort Lauderdale, Florida, USA.
14. Model Structure and Load Balancing in Optimistic Parallel Discrete Event Simulation, Tapas K. Som and Robert G. Sargent, Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS 2000).
15. A Unified Resource Management and Execution Control Mechanism for Data Flow Machines, Masaru Takesue, ISCA 1987:90-97.
16. Scalability and Potential for Optimization in Dynamic Load Balancing - Centralized and Distributed Structures, Wolfgang Becker and Jörg Zedelmayr, Mitteilungen GI, Parallel Algorithmen und Rechnerstrukturen, GI/ITG Workshop Potsdam, 1994.
17. Cooperative Load-Balancing System, M. Kudo and Y. Tozawa, Proceedings of the Seventh Conference on Artificial Intelligence Applications CAIA-91 (Volume II: Visuals), 259-260, Miami Beach, FL.
18. Laod Distribution: a Survey, Luis Paolu Peixoto dos Santos, Technical Report, UM/DI/TR/96/03, Departmento de Informatica, Universidade do Minho, Portugal.
19. High Performance Incremental Scheduling on Massively Parallel Computers- A Global Approach, Min-You Wu and Wei Shu, SC 1995.
20. Service Level Routing on the Internet, N. Anerousis and Gísli Hjálmtýsson, proceedings of Globecom'99, Rio de Janeiro, Brazil, December 1999.
21. Dynamic Load Balancing in Parallel Database Systems, Erhard Rahm, Proceedings of EURO-PAR'96 conference, LNCS, Springer-Verlag, Lyon, August, 1996, (Invited paper).
22. Distributed Cooperative Web Servers, Scott M. Baker and Bongki Moon, WWW8 / Computer Networks.
23. Managing Server Resources for Hosted Applications, Vikas Agrawal, Girish Chafle, Neeran Karnik, Arun Khirbat, Ashish Kundu, Johara Shahabuddin, and Pradeep Varma, (Patent Application pending), JP920010088, IBM India Research Lab.
24. Component Load balancing, Technology overview, Microsoft Application Center 2000.
25. Dynamic Load Balancing on Web-Server Systems, Valeria Cardellini et al, IEEE Internet Computing, May-June 1999.
26. An Architecture for Virtual Server Farms, Vikas Agrawal, Girish Chafle, Neeran Karnik, Arun Khirbat, Ashish Kundu, Johara Shahabuddin, and Pradeep Varma, Technical Report, IBM Research, RI01006, 2001.