COOPERATIVE LOAD SHARING AMONG LOOSELY COUPLED OR INDEPENDENT VIDEO SERVERS

Xavier Denis
Robert Duzett
ARRIS

Abstract

As video delivery systems are tasked with larger and larger content libraries, significantly larger streaming loads across increasingly broader geographies, and a wide diversity of client devices, the need for more dynamic scaling and virtualized provisioning of large on-demand resource pools is apparent.

This paper examines strategies and policies for enabling a logically-shared video delivery load across multiple independent but networked video servers, and provides insight into practical applications such as efficient and scalable models for the origin, mid-tier, or edge caching points of a Content Delivery Network (CDN).

INTRODUCTION

The fast growth of video traffic to computers, tablets and other mobile devices illustrates the profound changes affecting the ways in which people access video content. To compete with an increasing array of entertainment options, MSOs must harness the strength of their programming and work to facilitate access to it across the wide array of devices used by consumers.

These market forces are bound to have a significant impact on infrastructure and operations. Content providers have more options today to reach their target audiences. To keep the upper hand, MSOs need more efficient content ingest / packaging operations to enable fast capacity growth and quickly turn around assets that can be consumed on multiple platforms and client devices. As content continues to proliferate, consumption will increase, driving networks to near capacity. In fact, experts predict managed IP video traffic will quadruple over the next 4 years.

Instead of locally-dedicated servers and storage systems with bounded growth limits and limited load sharing, today's video loads call out for networked resource pools (processing, storage, and streaming resources) that can be easily deployed at discrete locations within a Content Delivery Network (CDN) architecture. These networked resource pools respond to changes in load levels and format mixes with agility, provisioning the resources of semi-independent and loosely-coupled servers and systems in an organic and efficient way.

This paper first discusses the requirements associated with next generation video delivery. After introducing a server pool architecture enabling multiple loosely-coupled

independent video servers to operate as one virtual load sharing system, we study the central issues of this proposed approach, including content allocation, load balancing and asymmetric load behaviors. Analysis is supported by simulations driven by real-world field data. In addition, we discuss the advantages of the proposed server pool architecture in terms of scaling / upgrading and review a practical application of the architecture to caching. Conclusions and key findings as well as topics suggested for further research are found in the final section.

VIDEO DELIVERY APPLICATION REQUIREMENTS

The challenges highlighted in the previous section imply new requirements to ensure efficient and scalable video delivery:

- Cost-Efficiency – The need for cost-efficient scalability puts unsustainable pressure on proprietary platform design models. Instead, video delivery solutions must leverage commoditized hardware to exploit the advantages of Moore's Law.
- Low latency and reduced network load – CDN network demands must be minimized to enable high quality delivery, especially for live video applications. Employed as a cache, the video delivery system has a role to play in ensuring low latency and reduced network load.
- Modular and agile scaling – There are multiple dimensions of resource scaling, including streaming capacity, content storage capacity, and ingest capacity. Streaming and storage provisioning requirements vary widely across different deployments and even across different hierarchical levels within a given network. For instance, the two core elements of a CDN, the origin and the cache, dictate nearly opposite sets of provisioning requirements, the former needing lots of inexpensive storage, the latter requiring high I/O performance and streaming density, but reduced storage overhead costs. In other words, the video delivery system architecture needs to support a provisioning model that is flexible while consistently/uniformly efficient, even for fairly asymmetric deployments and upgrades.
- Maintenance / upgrade agility – Upgrades must be straightforward, simple, and quick as possible, having as little impact on the existing system as possible, i.e. modular. Re-building, re-loading, or replacing elements of the existing configuration, in order to perform an upgrade, should be avoided.
- High Availability – The video delivery infrastructure must maintain constant service uptime. This implies redundancy and load balancing across separate resources with fail-over policies.
- Client device independence – modern video delivery platforms must offer flexible and dynamic resource provisioning mechanisms to transparently accommodate different delivery formats and varying workloads to a wide array of devices.

Today's video delivery applications demand increased IO performance, provisioning flexibility, scalability and reliability in addition to accommodating a

wide array of content access patterns and diverse file formats and sizes, all of which must be achieved with greater cost-effectiveness. In the following section, we define a flexible framework fulfilling the above requirements while offering great design freedom, including the ability to leverage commoditized hardware and standard networking protocols for optimal cost-performance.

ARCHITECTURAL SCOPE

The architectural scope of this paper encompasses a "server pool" approach to video delivery, in which the server pool is defined as a collection of loosely-coupled, semi-independent servers in a common network, with common reach-ability to video subscribers. Each server is a "node" of the interconnected delivery system. Both the available video content and the streaming load are distributed across the nodes.

The server pool architecture borrows the following key principles from Cloud Computing [1]:

- Shared resource pooling for increased utilization and efficiency
- Reliability by distributing load across multiple hardware instances and eliminating single points of failure
- Scalability by enabling capacity expansions through server instance additions without service disruption
- Maintainability by enabling upgrades and hardware maintenance without service disruption

While the descriptions and analysis in the sections below refer to "content" in general and thus seem to imply a stand-alone content library, a la CDN origin server, these concepts are equally applicable to a pool of cache content nodes. The scalability, redundancy, storage efficiency, and load-balancing efficiency of this solution can serve to optimize the cost-performance and operational efficiency of any appropriate location of a CDN, including the origin server, caching edge sites, or mid-tier caching or library sites. For simplicity of description and analysis, generic references to "content" and "objects" are used, with little mention of caching. However, near the end of the paper, after the core analysis sections, a "Caching Model" section describes methods for mapping these concepts onto a pull-thru cache pool.

Some general principles that are imbued in this architecture include:

- Virtualization – multiple nodes appear, and work together, as one. This applies to the content, whether as a library or as an aggregated cache, as well as the streaming resources and the node pool network as a whole. The pool concept is especially well-suited for exploiting shared end-user connectivity and pooled resources to efficiently aggregate and amplify the unified cache performance. All nodes in the pool have shared affinity with, or common reach-ability to, the video end-user. Policies designed to ensure maximum content heterogeneity across multiple servers' caches drastically increase effectiveness of the aggregate cache, thus positively affecting cache

hit performance at the edge and reducing traffic in the network [2]. Once the algorithms are in place that enable the servers in the pool to organically manage their respective resources, the aforementioned policies are easy to implement.

- Network and storage tradeoffs – inter-node communication, hierarchical network loads, and content propagation can all be reduced by judicial provisioning of additional content storage; and vice-versa.
- COTS – in support of the need for easy and large-scale modular resource scaling, using inexpensive and ubiquitous modules, the hardware server platform and underlying system software elements (OS) are assumed to be COTS and the network elements are standard commodity Ethernet devices. Furthermore, this COTS foundation encourages and enables the development of hardware-independent value-add software to implement the concepts introduced in this paper.

Architectural Objectives

- Balance - Balance streaming loads efficiently according to the capacities (content and streaming) of the various nodes. Establish rules and algorithms for provisioning nodes and resources, allocating content objects to nodes, and directing streams to nodes. Inter-node content movement to correct imbalances should be kept to a minimum.
- Scaling - Configure, scale-up, and upgrade overall capacities with minimum disruption to system operation, and asymmetrically (capacities of the various nodes may differ) if necessary. Establish policies and guidelines for adding nodes & capacity.
- Redundancy - Maintain services at rated capacities even in the face of a full node failure. Establish redundancy methods to fail-over lost content & streaming. Ensure:
    - Continued accessibility to all content titles
    - Continued full rated streaming capacity
    - In the case of a failed node in a caching pool, minimal network traffic devoted to content re-acquisition from the CDN origin server

With these objectives in mind, some key architectural elements are proposed to enable pool-based video delivery.

General Approach

The redundancy requirement calls for full rated streaming capacity and continued access to all content in the face of one off-line node, i.e. n+1 redundancy. In this, the architecture shares many of the goals of other RAIN (Redundant Array of Independent Nodes) architectures. This requirement is the most visible driver of the architecture's general methods, briefly described here:

- Store at least two copies of every video content object, with the copies on different nodes. Provision storage for this extra content. Dual-copy content allocation strategies are further described in the next section below, "Allocating Content". It should be recognized here

that, in the case of a caching pool, policies may be applied that dynamically vary the level of redundancy across the objects, some with no copies or 1 copy (relying on the origin server for backup) and some with 2 or more, depending on the popularity or streaming load of the object (see the Caching Model section later in this paper).

- Provision video streaming capacity across the system in such a way as to absorb one node going totally off-line. De-rate the streaming capacity of a node to account for the failover streaming capacity that must be reserved in case one of the other nodes fails. In other words, to cover the potential loss of one node, the system will ideally require no more than the streaming capacity of n+1 nodes to ensure an effective streaming capacity of n nodes. Capacity de-rating strategies and equations will be described in a later section, "Directing Streams".
- When a stream is requested, direct it to one of the nodes that has a copy of the requested object. Provisionally reserve an equivalent streaming load at one of the other copies, for possible node failover coverage. Selection of the streaming and failover nodes must consider the current streaming loads and provisional failover streaming reservations of the set of nodes with copies of the object. Stream-direction (node-selection) algorithms will be discussed below.

ALLOCATING CONTENT

In this architecture, content allocation entails replicating each content object on two nodes, being prepared to add a third copy dynamically as needed. This approach is in essence a heterogeneous distribution with limited (the minimum) replication, forming a judicial synergy of network and storage resources. Total provisioned storage is effectively 2 * rated content size, the minimum to meet the node failover redundancy requirement. One could extend this to the extreme and put a copy of all objects on all nodes, i.e. full content replication. This would maximize flexibility and simplicity in stream allocation and failover, as well as the ability to absorb streaming load asymmetries by stream direction only, but at the cost of significantly more storage, especially as the pool's node count increases.

So, again assuming only two copies per object, every object will belong to a specific content node pairing or "object group", which consists of all the objects whose two copies are allocated to the same pair of nodes. An object will belong to only one object group, but a node will intersect multiple object groups, one for each of the other nodes in the system. In a system of n nodes, there will be n(n-1)/2 object groups. In a graph of nodes, if one were to draw an arc between every pair of nodes, those arcs correspond to the object groups. This is illustrated in the diagrams here:

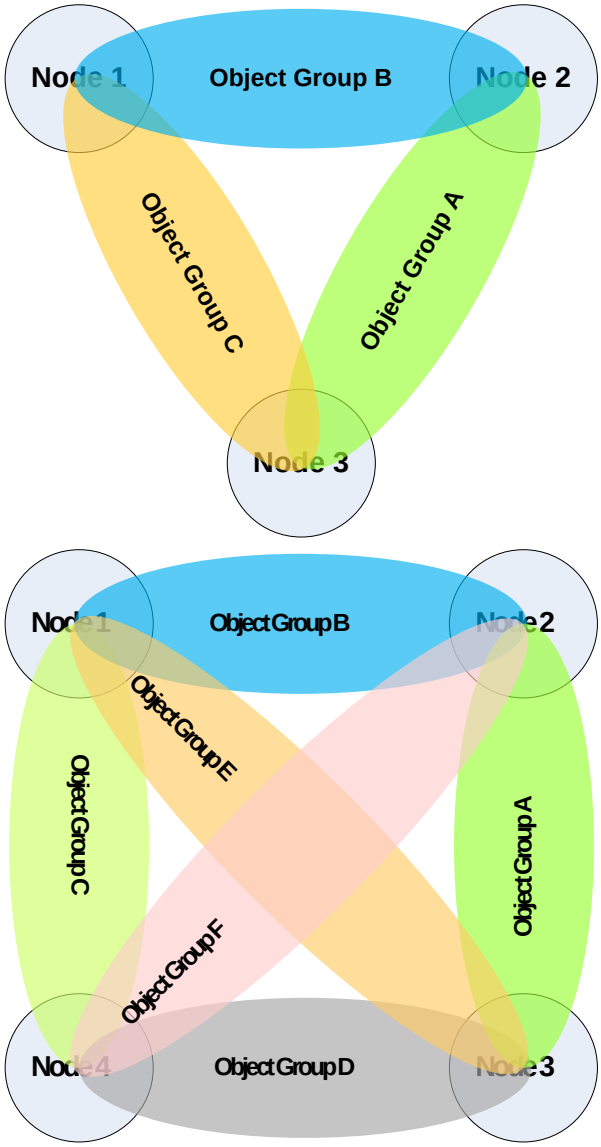


Figure 1 - Object group definitions for 3-node and 4-node server pools

Note that this object group concept can be extended to encompass a variety of node group sizings in the same system (see the "Exceptional Asymmetries" section for a description of 3-copy object groups). For now, the focus will remain on two-node object groups.

Content should be allocated across the nodes in as balanced a way as possible.

Ideally, all the nodes will end up with very near the same amount of stored content (weighted for relative capacities), and all the object groups will also be nearly equal in their storage allocation. A careful, even allocation of content objects across the nodes will lay a smooth, flat foundation for stream allocation – a sea of object copy pairs spread randomly across the array of nodes, ready for a streaming load and failover reserve to be carefully mapped onto it in as balanced a way as possible. Content should be allocated in a way that distributes objects to storage in an apparently random way, without regard to expected popularity, thus naturally mixing popular and less-popular objects within and among the object groups, maximizing the opportunity to absorb streaming hot spots and balance loads using the redundant resources provided (redundancy expands choice and flexibility). In effect, randomized but even content allocation tends to flatten the apparent content usage profile from the perspective of node and network utilization.

Another aspect of content allocation is content pre-placement. For the case of caching pools, pre-positioning of content may or may not be desirable or practical, depending on the specific implementation. Significant reductions in network loading have been shown for metadata-directed pre-placement of cache content. Regardless, one can pre-place none, some, or all of the content while following the allocation scheme mentioned above; and then place new content as it arrives in the same manner; or direct/re-direct streams for pull-thru to result in the same desired placements (more details on pull-thru approaches are given in the "Caching Model" section further below).

DIRECTING STREAMS

Stream requests should be directed to nodes in such a way as to minimize the de-rating of system streaming capacity, absorbing in an optimal way the failover streaming load of any off-line node as well as asymmetries in streaming demand (popularity hot spots). See examples of streaming load asymmetries in the table and diagram below. The streaming load asymmetries are expressed as the largest fraction of total system streams sourced from any object group (multiple "trials" are shown, in which the random elements of the content allocation mechanism are re-seeded). The data in the table was generated with simulations driven by real-world field data. More details on "absorbing asymmetries" are given in a section further below.

Asymmetry - Largest streaming load on an object group	3 nodes	4 nodes	8 nodes	16 nodes
Ideal (perfectly balanced)	.333	.167	.036	.0083
Trial #1	.363	.210	.103	.059
#2	.416	.224	.116	.059
#3	.383	.208	.085	.056
#4	.359	.215	.113	.075
#5	.423	.211	.086	.073

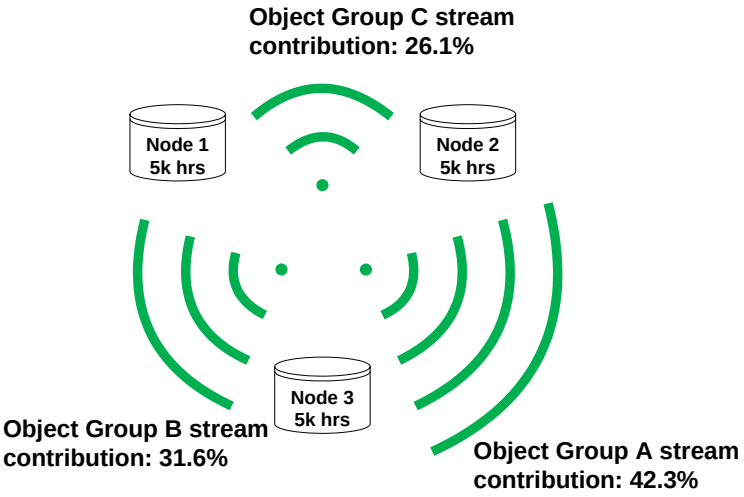


Figure 2 - Asymmetric streaming loads across object groups

When a new stream request occurs, a node must be selected to source the stream. Actually, two nodes are selected for every stream request – one to stream and one to provisionally reserve failover resources for the stream. Both of these nodes must have access to the content object requested and must be able to reach the appropriate transport network with the stream. If content has been allocated in the manner described in the section above, there will be exactly two nodes with copies of the desired object, so selecting one node automatically selects the other. Effectively, the object picks the node pair and the "stream director" merely decides which node of the pair will source the stream and which will shadow the stream for possible fail-over. These two nodes together uniquely describe the identity of the "object group" containing this object and others with their two copies on these same two nodes.

Since every object is uniquely assigned to an object group, the streaming load allocated to the object group at any given time is the aggregated streaming load at that time of all the objects of the group. In addition, an equivalent failover load is also allocated to the group. More specifically, a portion of the object group's streaming load is assigned to one of its associated nodes while the other portion is assigned to the other node in the pairing. The associated failover loads are apportioned to the opposite nodes of the pairing, so the total streaming-plus-potential-failover loads assigned from this object group to each of the two nodes hosting it are always equal.

Every node intersects a number of object groups, each of those groups being uniquely associated with this node and one of the other nodes. Therefore, the set of object groups of one node will overlap exactly one group belonging to another node, but each node's set is unique. Therefore, the total collection of object copies of one node are never the same as that of any other node. Likewise, the streaming loads and provisional failover loads of one node are unlikely to match those of another node.

The streaming capacity of a node must be sufficient to cover both its expected nominal streaming load and its worst-case allocated failover streaming load. At a given node and a given point in time, each of the object groups associated with that node contribute to the node a portion of the current streaming load of the object group and a complementary (corresponding to the other portion of the object group's streaming load) potential failover load. The same object group contributes the opposite loads to the other node associated with the object group. The worst-case total potential streaming load of a node at a given time is the **sum** of the actual streaming loads allocated to that node from all the object groups for that node, **plus** the **maximum** of the potential failover streaming loads allocated to that node from its set of object groups. The maximum failover load from among the object groups is used instead of the sum because only one of the nodes in the system is expected to be off-line at any one time and the worst-case scenario for this node is the failure of the complementary node of the object group that contributes the largest potential failover load to this node. So, for time t, the potential load l(t) at a node is given by:

$$l(t) = \text{total\_allocated\_streaming}(t) + \text{max\_allocated\_failover}(t)$$

and the required minimum streaming capacity that must be provisioned for that node is the maximum l(t) over the lifetime of the node's current configuration. A simple formula relating node capacity with the peak system streaming load and system size (node count) is given in the "Absorbing Asymmetries" section further below.

The objective in choosing one node over another to source a new stream is to maintain a balanced maximum streaming load across the nodes of the system, i.e. to match a node's worst-case load against its relative streaming capacity in the system. Since the worst-case streaming load, and thus the required capacity, of a node, is the total current streaming load plus the maximum current potential failover reserve, this is the metric that should be compared when selecting one node over

another to direct a stream. Selecting a node to source a stream based solely on the current actual streaming load of the candidate nodes will NOT result in a balanced system but will in fact lead to an extremely off-balance system that will not be able to fully fail-over the streaming load of a lost node. Nominal streaming loads will be balanced, but not the maximum potential load (i.e. after failover).

The graphs below compare two approaches to balancing loads, one based solely on the nodes' current streaming loads and one based on both streaming and maximum failover loads. Results are shown for a 3-node system, showing maximum streams and maximum streams+failover for all 3 nodes. The data was generated from simulations driven by actual field data.

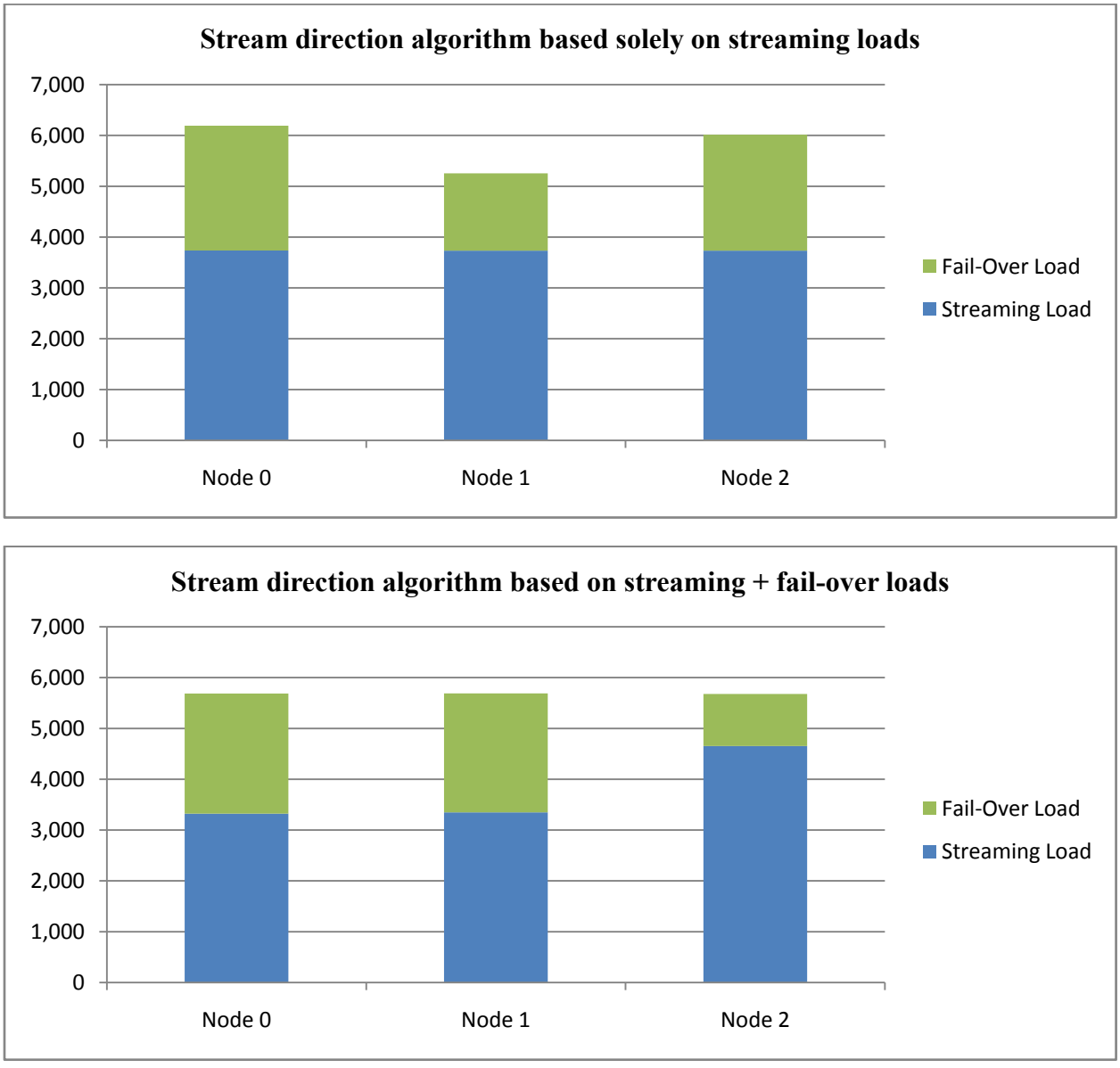


Figure 3 – Comparing load distributions with different direction algorithms

When only streaming loads are considered, the resulting maximum streaming loads are very well balanced, but total possible loads including failover are unbalanced and worst-case is quite high. When streaming+failover loads are considered, however, the streaming loads alone are unbalanced but the total load is well-balanced and lower. Streaming load alone means nothing when maximum potential failover streaming must also be reserved.

When an object is identified to source a new stream, that stream will be assigned to one of the nodes of the pair associated with the requested object's two copies. The node selected should generally be the one with the lowest current potential load, i.e. total streaming load plus maximum potential failover load. This is because the incremental streaming load will always translate completely to additional load on a node, while the incremental potential failover load may or may not add to a node's maximum failover load (a node's potential failover load from this object group may not be the current maximum for the node). So, the stream is directed to the node, of the pair, that has the lowest [streaming plus maximum failover] load, and the failover role for the stream is assigned to the other node.

ABSORBING ASYMMETRIES

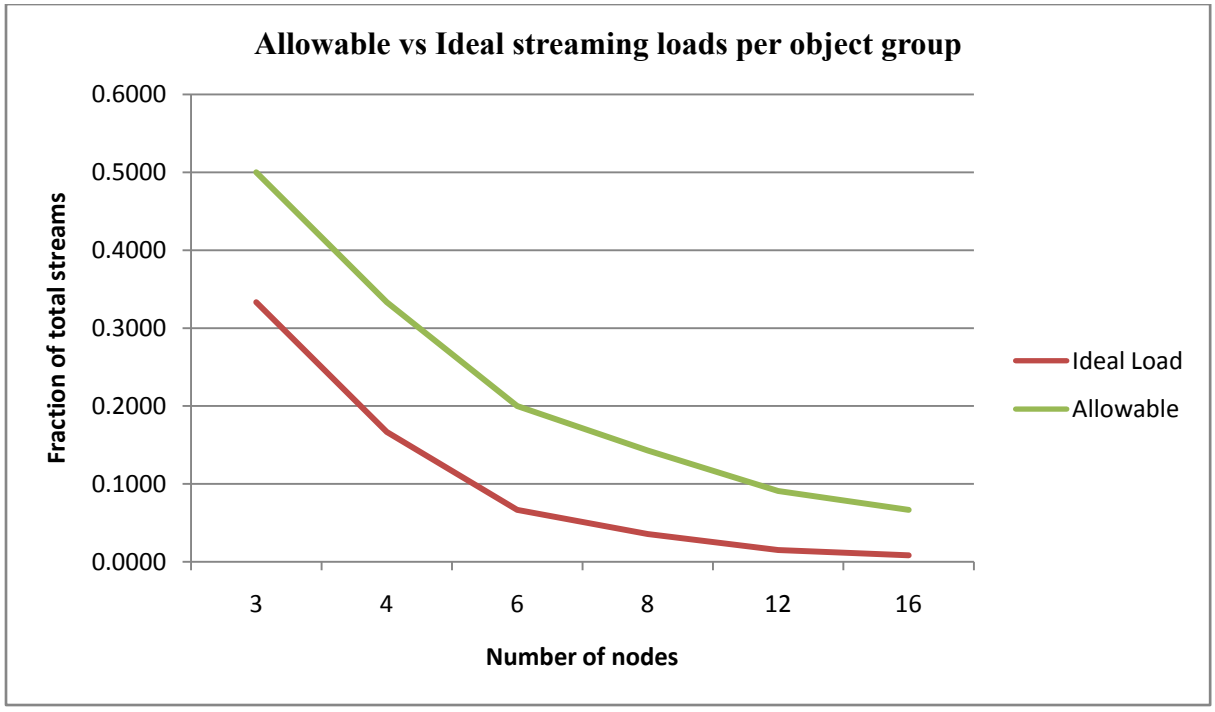
The asymmetry introduced by node failover has been addressed by the object storage and stream load redundancies described above. By storing two copies of an object on different nodes and accounting for failover streaming loads when de-rating a node's capacity and when assigning a stream, the possibility of a node failure is anticipated and provisioned for.

Note that the architecture's provisioning for content and streaming redundancy to cover a node failure also provides natural flexibility in stream allocation that will support efforts to balance uneven streaming loads across the nodes and object groups. Under most conditions, rules guiding content replication and the over-provisioning of streaming capacity will be sufficient to maintain balance in the face of dynamic load asymmetries (shifting popularity profiles) as well as to absorb a node fail-over, without having to move or adjust content. This has been verified by analysis and by simulation driven by real-world field data (just one example is shown in the graphs above).

In a system in which content objects and streams are allocated as described in the sections above, and given a peak total streaming load "S", the maximum streaming+failover load experienced by any of the nodes should nominally be 1/(n-1) * S (this equation simply represents the system streaming load spread across all the nodes but one, possibly failed, node; this is the de-rated capacity of a node). This per-node maximum will hold for a range of streaming load asymmetries described as follows: if each node of a system is provisioned to support at least 1/(n-1) * S streams, and the worst-case streaming demand on any object group of the system is less than 1/(n-1) * S, the nodes will fully absorb the streaming load as well as the failover load of any node failure. The determination of this upper limit to object group streaming load is based on the observation that each node of a pair must be

able to absorb the full streaming load of their associated object group because the group's streaming+provisional_failover load is double the streaming load and is evenly allocated to the two nodes. A streaming load, on an object group, greater than the capacity of either of its paired nodes is thus guaranteed not to be absorbable by the nodes. Note that a perfectly even distribution of streaming load would allocate 2/(n(n-1)) * S streams to each object group (there are n(n-1)/2 object groups (node pairings) in a system of n nodes). This means the maximum absorbable object group load is n/2 times the perfectly even (ideally balanced) load. See the table and diagrams below for streaming load asymmetry ranges for various node counts.

#nodes (n)	#object groups (arcs connecting node pairs) 2/(n(n-1))	Minimum Rated Node Capacity – fraction of system streams 1/(n-1)	Range of allowable max streaming load per object group n(n-1)/2 – 1/(n-1)	Allowable/Ideal ratio n/2
3	3	1/2	1/3-1/2	1.5
4	6	1/3	1/6-1/3	2.0
6	15	1/5	1/15-1/5	3.0
8	28	1/7	1/28-1/7	4.0
12	66	1/11	1/66-1/11	6.0
16	120	1/15	1/120-1/15	8.0



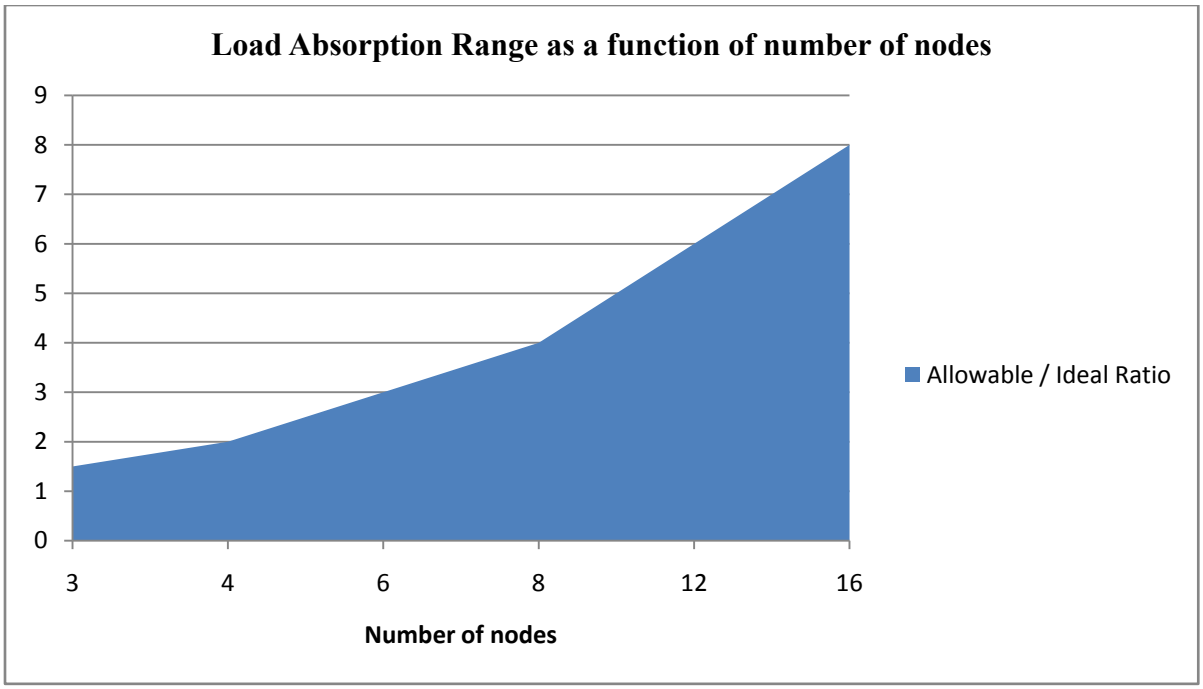


Figure 4 – Headroom for streaming load asymmetries

Simple n+1 provisioning, combined with 2-copy content provisioning and efficient allocation, is sufficient to absorb most practical asymmetrical situations if the stream direction algorithm is also effective (balance streams + max-failover, NOT just streams).

Given this most basic provisioning, a smart content allocator, and a smart stream director, how much asymmetry can be absorbed? The table and graphs above show the theoretical bounds of asymmetry for various system sizes. Below is a graph containing some examples of various load asymmetries, from simulation models driven by actual field data. Shown are the streaming loads of the object groups of a 4-node system, these loads all clustered around the ideal balanced load level and all below the absorbable limit calculated above.

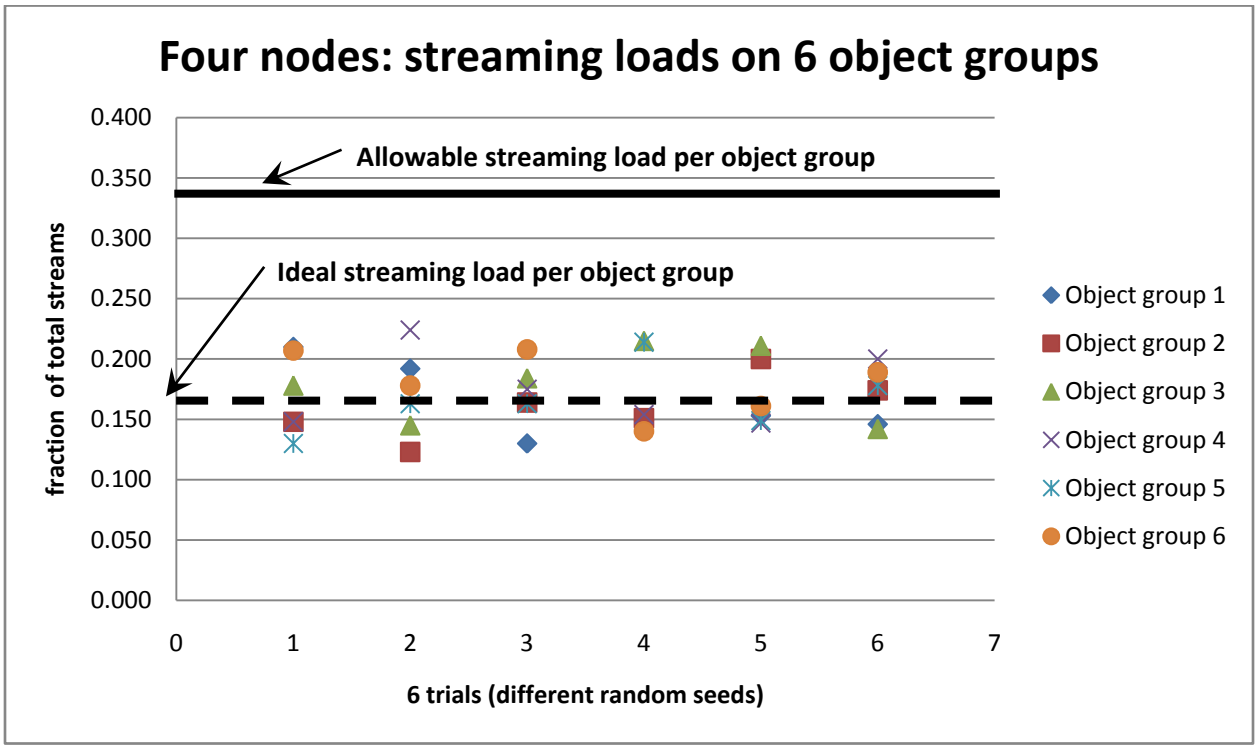


Figure 5 – Absorption of asymmetric streaming loads

Normal everyday hot spots are generally exhibited by a group of popular objects that are scattered randomly across the object groups and so tend to exhibit themselves as minor imbalances in demand. These migrate over time, exhibiting normal fluctuations in user demand but staying within the absorption range defined above. However, shorter-term, faster-ramp and higher-magnitude spikes in demand for isolated objects can also occur. If these are not too severe and/or they occur while no nodes are off-line, they are also generally absorbed successfully.

The exceptional scenario is the sudden demand for an uber-popular object that soaks

up a significant fraction of total streaming capacity for some period of time, i.e. the so-called "super bowl" scenario, especially if, unlike the Super Bowl, it's unplanned and unpredictable. Depending on the number of nodes in the system, an object group that suddenly accounts for 1/4th, 1/3rd, or ½ of all streams because of one or two super-hot objects could easily exceed the bounds given in the table above. This is when dynamic propagation of extra object copies to other nodes becomes important.

Note from the tables and graphs above that, although the ranges of relative allowable streaming asymmetry is much higher for large node-count systems, the absolute maximum loads are much smaller than for small node-count systems. Thus, as systems grow to higher and higher node counts, they are actually more vulnerable to isolated uber-popular objects.

Be aware that the inherent absorption range of the node pool can be expanded to virtually any reasonable level by further de-rating the streaming capacity of the nodes. This is an alternative that can be traded-off against more content redundancy (more copies to begin with or more extra-copy dynamic object propagations). Another alternative is to go the other direction and simply plan on absorbing a fixed but reduced level of asymmetric and/or failover loads, as one's risk tolerance dictates, and accept that some capacity may be unavoidably curtailed (or provisioned as extra load on the origin server and the intervening network).

Exceptional Asymmetries

The system must be able to handle asymmetries that exceed de-rated capacities. As indicated above, the streaming load on an object group should not approach or exceed 1/(n-1) * S. When imbalances or load fluctuations overtax an element of the existing configuration, dynamic adjustments must be made to rebalance the load. The approach taken by the proposed architecture is to dynamically propagate or pull-thru additional copies of the problematic object to nodes with unused content & streaming capacities.

Adding a third copy of an object creates an effective triangle of nodes and thus three node-pairings to which the object's streaming and provisional failover assignments can be made. This in effect creates a new object group of 3 nodes tiered above the two-node object groups triangulated by those nodes. It gives the stream director three possible node pairings from which to choose when assigning a new stream, rather than just one. The additional node pairs are available to absorb the excess loads being experienced by the original 2-copy object group.

To accommodate a reasonable 3rd-copy capability, the nodes of the pool must be provisioned with a fractional increment of unallocated content storage. Exceptional asymmetries are generally caused by just a few objects, so the required incremental capacity is relatively small.

SCALING AND UPGRADING

A prominent feature of a networked server pool architecture is its promise of easily

scaling up or scaling down the pool's resources by adding or removing nodes or modules. This eases the operational load and costs of system capacity upgrades & maintenance, all with minimum disruption to active operations.

Adding new content objects to the system, when there remains existing storage capacity, is straight forward. The content allocation method described earlier in this paper should continue to work as long as no node fills its storage nor steals from 3rd-copy reserve. If, however, storage is full or near full, additional storage should be added to the system. One could add incremental storage, i.e. independent storage volumes or shelves, to each (or some) of the nodes, or one could add additional node(s) to the existing ones. Either way, the new storage capacity should be evenly and smoothly integrated into the system by, for example, randomly selecting existing object copies to be migrated to the new storage, and disabling the old copy from further stream allocation and ultimately deleting it. This should continue at an acceptable pace until the storage utilization is once again even (by weight) across the volumes.

To add streaming capacity, it is generally easier to add standard nodes to the pool than to add CPU and IO capacity to a node.

While arbitrary resource asymmetries cannot always be well-balanced or efficiently exploited, systems can be incrementally scaled-up with nodes that are provisioned with resource capacities different from those of the original nodes.

CACHING MODEL

This node pooling architecture can be applied to a cache pool as well as it can to a content library pool. In a pool of pull-thru cache nodes, for example, cache content could be provisionally allocated by directing a primary stream pull-through at one node and a secondary pull-through (for potential failover coverage) at another, letting the cache logic and state of the individual nodes determine whether and how long the content stays in the cache. The stream's failover node does everything the streaming node does, including provisionally reserving the streaming bandwidth, except it doesn't actually stream the content. Future stream requests for the same object are directed at the same pair of nodes with both nodes hitting and/or updating their caches accordingly and one being chosen to stream while the other provisionally reserves bandwidth for failover.

While some objects will take up storage space in two caches, this is a minimum redundancy ensured by the disciplined content allocation mechanism. If the streaming activity of an object is sufficient to cause it to naturally hold a place in the two caches, the potential impulse load on the network and other nodes caused by a failed node will be reduced, because another copy of the object is already cached. On the other hand, the limited redundancy approach of the content allocation scheme actually maximizes the uniqueness and heterogeneity of content across the caches, thus improving the cache efficiency of the server pool over ad-hoc methods that allow caches to all pull from the same large library .

Note that policies may be applied that vary the level of redundancy across the objects, some with no copies or one copy (relying on the origin server for backup) and some with two or more, depending on the popularity or streaming load of the object.

Another approach would initially assign a single node (and provisionally its cache) to an object and its associated streaming load, until the demand for that object warrants an additional caching node to offset the load and provide valuable failover capacity (again, avoiding an impulse load on the other nodes and the upstream network if a node fails).

CONCLUSIONS, IMPORTANT FINDINGS

- An efficient server pool architecture can be effectively applied to optimize the cost-performance of any location in a CDN, including the origin server, a caching edge site, or a caching mid-tier site.
- Storage costs can be reduced significantly by provisioning content storage (whether library or cache) for minimum inter-node redundancy (i.e. 2 copies). Even this limited redundancy provides the stream director with significant headroom and flexibility for absorption of both failover demand and asymmetric streaming loads (hot spots). Exceptional events are then sufficiently handled by propagating extra copies of selected objects.
- A careful, even allocation of content objects across the nodes will ensure a flat foundation for stream allocation, minimizing the effects of streaming load asymmetries occurring on top of the content pool.
- Provisioning for an off-line node (n+1 redundancy) is not sufficient in itself to ensure smooth or successful failover. Load balancing and stream direction logic must also come into play to anticipate, and allow for, a worst-case node loss (instantaneous demand spikes), not just to balance current streaming loads. Failover allowances cannot be made arbitrarily at the system-wide level nor by a fixed amount applied equally to all nodes globally. Appropriate allocation levels for streaming and failover will be node-specific, dynamic, unpredictable, and highly variable.
- Re-active content propagation can be avoided or minimized with smart provisioning. Relying solely on intelligent but straight-forward provisioning guidelines and stream direction algorithms, a multi-node pool can be statically configured and provisioned to maintain optimum balance while absorbing node failover and/or significant demand asymmetries, with minimum redundancy and cost. A minimum of dynamic churn (e.g. content propagation, and other dynamic load balancing methods) is required to handle exceptional outlier cases.
- The role of a stream director is simply to decide which node will source a stream vs which will provisionally reserve fail-over bandwidth, but this decision has major impact on load balance and resource utilization. Significant streaming load asymmetries are intelligently absorbed while piggy-backing on the

minimum node failover provisioning (both content & streaming).

FURTHER RESEARCH

- 3rd-copy and 4th-copy dynamics
- Asymmetric upgrades, including >2 different capacities in the server pool
- Command, control, & communications to enable & support decision-making
- N+2 redundancy
- Other affinities & dimensions thereof, including network loads & capacities
- Bit-rate striping
- OTT video cache-ability and how to increase it?
    - How much is currently cache-able on a network?
    - How different is it within an operator's network?
- Transport & storage cost models

REFERENCES

1. *NIST.gov – Information Technology Laboratory – Cloud Computing Program (http://www.nist.gov/itl/cloud/index.cfm)* – Retrieved 5/2/2011
2. *Managed CDN – Optimizing the Behavior of Hierarchical VOD* - Robert Duzett (ARRIS), Jeremy Craven (ARRIS)