[TMK+09] Gareth Tyson, Andreas Mauthe, Sebastian Kaune, Mu Mu, Thomas Plagemann. **Corelli: A Peer-to-Peer Dynamic Replication Service for Supporting Latency-Dependent Content in Community Networks.** In: Reza Rejaie, Ketan D. Mayer-Patel: Multimedia Computing and Networking (MMCN '09), Multimedia Computing and Networking (MMCN), SPIE and IS&T, January 2009 Suite

Corelli: A Peer-to-Peer Dynamic Replication Service for Supporting Latency-Dependent Content in Community Networks

Gareth Tyson[a], Andreas Mauthe[a], Sebastian Kaune[b], Mu Mu[a] and Thomas Plagemann[c]
[a]Computing Department, InfoLab21, Lancaster University, Lancaster, UK
[b]Multimedia Communications (KOM), Darmstadt University of Technology, Germany
[c]Department of Infomatics, University of Oslo, Oslo, Norway

ABSTRACT

The quality of service for latency dependent content, such as video streaming, largely depends on the distance and available bandwidth between the consumer and the content. Poor provision of these qualities results in reduced user experience and increased overhead. To alleviate this, many systems operate caching and replication, utilising dedicated resources to move the content closer to the consumer. Latency-dependent content creates particular issues for *community networks*, which often display the property of strong internal connectivity yet poor external connectivity. However, unlike traditional networks, communities often cannot deploy dedicated infrastructure for both monetary and practical reasons. To address these issues, this paper proposes Corelli, a peer-to-peer replication infrastructure designed for use in community networks. In Corelli, high capacity peers in communities autonomously build a distributed cache to dynamically pre-fetch content early on in its popularity lifecycle. By exploiting the natural proximity of peers in the community, users can gain extremely low latency access to content whilst reducing egress utilisation. Through simulation, it is shown that Corelli considerably increases accessibility and improves performance for latency dependent content. Further, Corelli is shown to offer adaptive and resilient mechanisms that ensure that it can respond to variations in churn, demand and popularity.

Keywords: Community networks, peer-to-peer, replication, caching, streaming, self-organization, autonomous systems.

1. INTRODUCTION

Caching and replication has been an effective approach for lowering overhead and increasing accessibility and performance in distributed systems for a number of years. It has been used in a range of environments including web page distribution [36], multimedia distribution [5] and distributed object location [26], helping to ensure that the Internet remains robust and scalable. Large scale multimedia distribution and video streaming, in particular, can gain significant advantages from effective caching and replication due to their high bandwidth, low latency requirements. Poor provision of such facilities has been observed to create significant issues for ISPs when providing video content [3]. This has led to a number of projects looking at effective methods of replicating and caching video content in the Internet [15][16][31]. Whilst these approaches have been largely effective, the problem still remains that it is often infeasible to deploy such infrastructure in dynamic, small scale environments that are incapable of hosting dedicated caches. One interesting example is community networking [7][8], a growing area in which geographical communities of equal users are connected through an infrastructure that has the properties of strong internal connectivity alongside poor external connectivity. Common examples of this are wireless networks, wired communities and rural mesh networks. The effective provision of internal caching and replication for such communities is therefore critical to their performance.

In such environments, all users are seen as equal which creates significant issues when deploying and financing caching infrastructure since resources are usually controlled and distributed amongst all members of the community. It therefore becomes impossible to support dedicated server-based caching and replication services. However, it is still evident that significant gains in performance can be achieved if community environments were capable of deploying their own caching and replication services. Due to their unique constraints, a lack of such provision severely limits their capabilities when dealing with the low latency, high bandwidth requirements of video content. The need to repeatedly fetch information from external sources (e.g. origin servers and externally deployed peer-to-peer streaming services) therefore results in *i) poor performance* – often channel zapping and buffering times are unacceptable, *ii) higher overhead* – frequent utilisation of egress points becomes necessary resulting in congestion and lower available bandwidth and *iii) higher costs* – many communities are required to pay for their egress bandwidth utilisation making external communications extremely undesirable.

This paper proposes Corelli, a peer-to-peer alternative to dedicated video caching, designed for deployment in community environments. Corelli, like existing server based systems, can be utilised alongside any supporting video streaming service. Corelli's uniqueness, however, comes in its ability to be dynamically instantiated, managed and scaled by communities of users without administrative intervention, monetary investment or dedicated resources. Instead, Corelli autonomously manages itself by constructing and adapting caches using the resources of community members. This means that replication and caching infrastructure can be dynamically constructed to match the demand observed in individual communities. Through this approach, Corelli can allocate resources in a highly responsive manner to provide extremely low latency internal access to users without the necessity to endure lengthy and expensive interactions with external systems and networks. This allows communities incapable of deploying dedicated infrastructure to collectively improve their own performance whilst reducing overhead.

The paper is structured as follows; Section 2 provides a background to the research area. Section 3 introduces Corelli and outlines its design. Section 4 then evaluates Corelli based on a number of factors; primarily, Corelli's ability to adapt to changes in demand, content popularity and churn. These factors are then qualified against the overhead measured in the system. Section 5 then provides a performance evaluation to investigate the utility of operating replication in this manner. Lastly, Section 6 then concludes the paper, highlighting areas of future work.

2. BACKGROUND AND RELATED WORK

There has been an extensive body of work carried out in the fields of caching, replication and video streaming. *Caching* is the process of passively storing replicas of content as it passes through an ingress point, whilst *replication* is the process of directly copying content between points. Replication can either be passive or active; *passive* replication occurs when a host desires a piece of content and therefore downloads it. This benefits the system by creating replicas but is done out of the user's self-interest. Conversely, *active* replication occurs when the system actively copies content between points; this occurs regardless of whether or not these hosts desire the content.

We define a community network [8] as a geographically grouped set of peers that cooperate as equals. As a community, these members then share any available egress connections. One example of this is a rural village consisting of wireless Ethernet enabled houses but no broadband infrastructure. Such villages struggle to access conventional wired broadband services and therefore utilise point-to-point wireless connections to the Internet instead e.g. [7]. To enable this, an ad-hoc community is built between the houses in the village, providing community access to the out-band connection. This results in a cluster of strongly connected nodes sharing a restricted egress link (e.g. to the Internet).

A number of requirements can be derived from the unique nature of multimedia content provision in community networks. Unlike server-oriented approaches possessing high resources, it is important to be able to dynamically adapt the system to ensure effective resource utilisation. Further, this adaptation must be able to respond to variations in churn, demand and requirements. This allows a variety of potentially divergent communities to be serviced by Corelli. Importantly, replication must also be highly responsive since caching and active replication is only beneficial early on in the content's lifecycle as later requests can often be serviced through passive replication in the community. It must also be possible to dynamically construct (or remove) caches in communities. By this, communities with large variations in demand can easily utilise caches when necessary, allowing dynamic placement. Importantly, the system must also be able to provide low latency access to content with sufficient bandwidth to satisfy the encoding rate. Without this, users will witness extended buffering times and poor channel zapping. Also, due to the decentralised nature of communities, caches must be capable of autonomous self-management without the need for individuals in the community to perform the role of administrators. Similarly, cache construction must not have associated monetary costs, as the nature of communities makes it difficult to apportion costs or retrieve monetary contributions. Importantly, the mechanism must also be efficient, taking into account fair resource utilisation and load balancing. Therefore traditional caching cannot take place as it would be necessary to route all content through the cache, leading to a significant bottleneck. Instead, active replication must be utilised, ensuring only popular content is replicated onto the cache. Lastly, the approach must not be restricted to individual streaming systems since it must be possible to replicate content from a range of providers.

Current caching systems are primarily based on server-oriented technology [5][36]. These systems generally passively inspect protocols such as HTTP and RTCP to store content. However, they are severely limited by their passive role in the system, which prevents them from gaining intelligent insights into trends and other system characteristics. For example, they are incapable of interpreting replication rates in communities to exploit peer-to-peer distribution; this means that they can actually have a detrimental effect when working against other oblivious mechanisms. As an

alternative to caching, *replication* systems have also been proposed [31]. These dedicated servers, however, are inflexible and are statically placed in the network meaning that they struggle to deal with dynamicity in demand [32]. To alleviate this, distributed mechanisms have been introduced. Systems such as [12][19] interconnect multiple servers to improve performance whilst alternative mechanisms attempt to increase scalability and resilience e.g. [32].

Server based approaches offer simplistic management, however, their static nature means that they struggle to adapt to different scenarios. Further, their poor scalability and expense make them undesirable in large scale systems. In response to these issues a number of peer-to-peer approaches have been proposed. Cooperative networking [22] has been the focus of much study. Cooperative caching architectures such as SHARK [2] allow peers to augment existing file servers to improve performance. Cooperative video streaming and caching has also been investigated e.g [16][35]. Hybrid approaches such as PROP [15] reduce the role of centralised instances although dedicated resources are still necessary for system operation. Squirrel [17] and Coral [13] perform cooperate caching by utilising fully distributed object lookup [26] to locate replicas of web content. However, they deal with small, latency tolerant objects and cannot subsequently be ported to video distribution. Larger scale systems such as Flashback [11] and Overhaul [23] have also been proposed to augment web servers by utilising clients during periods of high demand. Such approaches, however, do not focus on reducing external communications or minimising latency; instead the focus is on dealing with flash crowds and improving scalability. This makes them inappropriate for provisioning video in communities that are restricted by their egress connectivity. Work such as [20][21] deals with the concept of *replication groups*; these are groups of nodes that cooperatively replicate content for each other. Corelli communities can be considered as a type of replication group, however, Corelli extends this concept to involve a peer-to-peer role-based infrastructure that specifically aims at high-bandwidth provision of latency intolerant media. A variety of replication algorithms for these systems have been studied; these have included studies into replication based on objects sizes, requests rates and space constraints [40]; peer-to-peer replication of objects [18]; and video replication [35]. These have been considered in the design of Corelli, however, our work focuses on the autonomic aspects of the infrastructure rather than the replication algorithms. Alternate approaches such as Pastry [26] and PAST [27] use more structured replication mechanisms. Though these systems are aimed at supporting individual applications (e.g. persistent storage) therefore preventing them from being used with streaming services. Further, their adherence to structured replications makes them too inflexible to be deployed in divergent communities. A number of peer-to-peer streaming systems also utilise end hosts as passive replication points. Examples include ZigZag [34] and CoolSteaming [39], however, these systems are not directed at community environments. This is due to their requirement for external communications making it necessary for peers to access nodes outside the community during bootstrapping and data exchange.

3. CORELLI DESIGN

Based on the requirements and motivation outlined in the previous section, Corelli has been designed to offer low latency content access in networked communities. Corelli caches are dynamically built from a subset of high capacity nodes in the community. These peers work in cooperation to monitor request trends and pre-fetch popular content early on in its lifecycle. When a client node operating in the system wishes to access a video, it issues a request through its community's Corelli cache; if the cache contains the requested object it is accessed locally. Further, Corelli's application level position in the distribution system allows it to gain insights not possible in a passive generic cache. This therefore allows Corelli to monitor replication levels in the community. This results in a two tier caching architecture in which requests can be easily redirected to other client nodes to improve performance and resilience.

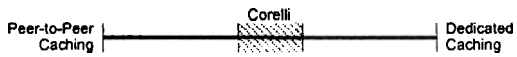

Figure 1 Location of Corelli on the Spectrum of Distribution

Corelli's *behaviour* can therefore be considered to lie between centralised and peer-to-peer technologies, shown in Figure 1. Corelli caches are constructed in a distributed manner, however, the use of high capacity nodes creates an autonomous cluster that can perform the supplemental role of a traditional caching server. Unlike traditional cooperative networking, this protects lower-capacity peers from the act of replication. This contrast further provides Corelli with a responsiveness, flexibility and ease of management not possible in conventional peer-to-peer caching approaches (e.g. [17][27]). Therefore Corelli can easily achieve tasks such as interchanging caching algorithms, adapting behaviour and monitoring community trends. This section covers the instantiation, adaptation and management of the Corelli caching and replication system alongside the caching and replacement algorithms utilised.

3.1 Joining the Corelli System

When a new node joins the system it is necessary for it to first locate its community's Corelli cache(s). Due to the diversity in potential community networks it is not efficient to strictly define a single discovery mechanism. This is therefore abstracted away from, allowing any mechanism to be used. This flexibility allows the most effective discovery mechanisms to be utilised for each environment. For example, JESA [25] might be used in a wireless community whilst Meridian [37] might be used in peer-to-peer community. This diversity can be easily managed through middleware such as ReMMoC [14] which supports the utilisation of multiple discovery mechanisms.

Once the cache has been located, the new node registers itself with it. This involves informing the cache of the objects that it possesses so that the cache can calculate how well items are replicated in the community. If the new node also has the capabilities to become a caching peer then this is also registered with the cache so that its resources can be utilised if necessary. The minimum requirements for caching peers are currently 128KBps connectivity, alongside 1GB of storage. It is also possible for the join request to be rejected by the cache if it is overloaded. This occurs if all the available caching resources are currently fully utilised. If no alternative accepting cache is located and the new node is capable of acting as a caching peer, it initiates itself in the role. This node then publishes its caching services through its community's service discovery mechanism.

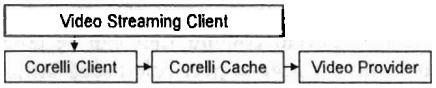

Figure 2. Corelli Request Redirection

Once this process has completed it is necessary for client nodes to route their content requests through their community's Corelli cache, shown in Figure.2. This must be done transparently for the video provider, to allow generic compatibility. A client middleware therefore offers a local proxy interface to applications which then forward RTCP/RTSP requests through it. A local stream is then established between the middleware and the application transparently whilst the middleware contacts the Corelli cache and any external content services (e.g. requesting chunks from BitTorrent [4]).

3.2 Cache Adaptation

During periods of high utilisation a single caching peer will become unable to service all requests. To remedy this, Corelli caches can dynamically adapt by enlisting the assistance of local peers to build a distributed cache. To select new peers, Corelli takes into account both peer resources (i.e. storage and bandwidth) and burn-in characteristics [10]. This dictates that only nodes that have been a member of the system for longer than m minutes are eligible as caching peers. Through this, the churn level in the cache is considerably reduced. This process therefore creates a reliable virtual cache that contains multiple peers working in cooperation. There are two occasions in which a cache might wish to expand: *i)* when it requires increased storage capacity and *ii)* when it requires increased bandwidth.

The need for increased storage capacity is signified by the cache reaching its capacity. In this event, content can either be ejected from the cache or the cache can expand. This decision is made using trend analysis. The hit rate is monitored throughout the cache's existence. If previous expansions have shown to increase the hit rate by an adequate amount ($e\%$) then the cache expands. If the cache has not previously expanded, the trend analysis is based on a previous hit rate of 0% which, in practise, always results in expansion. If the trend analysis indicates that cache expansion has ceased to offer improved hit rates then free space is created by ejecting content. An alternative situation is requiring increased bandwidth (this occurs during periods of high demand). If a cache reaches its provisioned bandwidth saturation for longer than three contiguous requests, it enlists the assistance of a new caching peer. The new peer then obtains its delegated chunks from the community (existing caching peers and clients) and begins hosting them.

As well as expanding the cache it is also possible to reduce its size. This allows resources to be freed for utilisation elsewhere during periods of disuse; this is done through churn. The decision to not replace a lost peer is made through trend analysis; if, based on recent activity, the loss of one caching peer is shown to lower the hit rate by less than $c\%$ then a failed peer is not replaced. The remaining caching peers then remove their least popular content, replacing it with the most popular content from the failed peer. This, however, is only performed if current demand can be adequately serviced by the remaining peers without exceeding 50% bandwidth utilisation. Generally, the contraction of a cache is a rare event as it is better to spread the load over a larger number of peers; this is achieved by setting c to a low percentage.

3.3 Cache Management

Once a cache becomes distributed over multiple peers it also becomes necessary to make distributed decisions. One possibility is to designate a single peer as a *cache coordinator*. This, however, is an undesirable solution due to the inherent vulnerabilities of a centralised entity. However, it is similarly undesirable to have fully distributed management as this can create high levels of overhead and complexity. Instead a hybrid approach is taken using *roles*. This separates management functionality into components based on certain roles that must be fulfilled. These *role components* can then be hosted on either a single peer or alternatively multiple peers. These then transparently interoperate through remote procedure calls. This allows different areas of functionality to be distributed in different ways depending on individual requirements. Six roles are identified in Corelli, outlined in Table 1.

Table 1. Role Components in Corelli

Role Component	Responsibilities
Caching Manager	Executes caching/replacement algorithms
Storage Manager	Decides which peers to store chunks on; indexing content in the cache
Adaptation Manager	Executes adaptation algorithm, deciding when to expand/contract
Request Handler	Receives content requests from client peers
Distribution Handler	Provides chunks to clients upon request
State Manager	Provides persistent, retrievable state access to all caching peers

Initially, all role components exist on a single peer; however, as the cache expands, these role components are deployed onto the other members of the cache. The *Caching Manager*, *Storage Manager*, *Adaptation Manager* are all hosted on single peers. By hosting these complex algorithms on single peers, overhead is reduced and efficiency improved. To ensure resilience, however, peers hosting these role components are monitored; on the detection of a failure, another peer is randomly elected to replace it. To allow this process it is necessary to replicate state information to ensure the failure of a peer does not remove important state data from the cache. To remedy this, a *State Manager* component exists on every peer in the cache and periodically, probabilistically replicates its node's state information to other State Managers (hosted on other peers). This ensures that up-to-date replicas of state information always exist in the cache. Therefore, upon detecting a failure, the new node hosting the role component broadcasts a request for recovered state information. Any remote State Manager component possessing copies of the information then responds. Finally, all the caching peers also host the *Request Handler* and *Distribution Handler* components; this allows all the caching peers to store content and act as access points for client nodes. This is done due to the comparative simplicity of these roles. This therefore allows them to be fully distributed amongst all the caching peers.

3.4 Content Placement Strategy

An important issue is how content is stored on the cache. Fair storage and bandwidth distribution is of the uppermost importance, as the restricted resources of the caching peers means that poor load balancing will result in significantly lower performance. To this end, a striped RAID 0 [24] placement strategy is employed where every object is broken up into chunks and then placed consecutively on caching peers. This approach allows storage and bandwidth loading to be evenly distributed amongst caching peers. Further, it allows client peers to download from multiple caching peers [4] to support the high bandwidth requirements of video provision. Such an approach has been selected over redundant strategies (e.g. RAID 5) for two reasons; firstly, sacrificing large space for parity data can have a significant effect on performance and secondly, because passive replication on clients allows data to be easily restored from the community.

Due to the distributed nature of the cache it is possible that different caching peers will have different distances from the client therefore returning varying quality levels. The strong internal connectivity of the community, however, alleviates this issue. Further, the use of multi-source distribution means that latency is measured as the average latency between the client and all the caching peers. This is because the client middleware combines all the chunks into an individual RTP stream provided to the application. One further issue is that the addition or loss of a caching peer requires data in the cache to be redistributed. When this occurs a replacement peer is located using the expansion process. This peer then downloads the chunks that existed on it predecessor from local replicas in the community (if available).

3.5 Caching and Replacement Algorithms

The primary difference between caching in Corelli and conventional caching is that as content becomes more popular it becomes unnecessary to maintain the content in the Corelli cache. This is because, as popular content becomes passively

replicated, sufficient sources exist within a community to handle the demand. Thus, Corelli's approach is a combination of both caching and replication. All *requests* are routed through the community's Corelli cache in a similar way to traditional caching. It would be inefficient, however, to have all *data* routed through the cache; therefore, only popular content is actively placed on the cache, creating high bandwidth, low latency sources early in the content's lifecycle.

Content is replicated onto the cache after *n* request for an item; this is termed the *caching point*. If the cache has reached capacity (and it has decided not to expand) an item with the least priority is removed to free up space. Priorities are defined by the priority list; this is an ordered list of importance for all the items in the cache. Two properties are taken into account when constructing and maintaining the list: *i)* the number of requests for each object and *ii)* the number of community replicas. Each object in the list is grouped with other objects that have received the same number of requests. These groups are further ordered by the number of local replicas available in the vicinity. Therefore, if there is contention over which item to remove based on the number of requests received, the decision is based on the number of local replicas available. Objects with a high replication rate are removed before objects with lower replication rates. To account for changes in popularity, request counters also age; after an object has been cached, every hour its counter is halved. This ensures items do not remain in the cache after their popularity has ceased.

Corelli does not store small objects as these objects are quickly replicated; nor does is store large objects (e.g. entire movies) as this would quickly exhaust resources. To remedy this, prefix caching [30] is employed by which the high demand periods of videos are cached. It is observed in [9] that 77-79% of video sessions last under 10 minutes. Further, [38] found a correlation between the popularity of content and the length of time users watch it. It was demonstrated that the average viewing time for the most popular items of content is approximately 10 minutes. Similarly in content such as football matches users are inclined to skip to high points such as goals [6]. This indicate that the use of *high point* caching can be employed to allow nodes to gain a low latency 'head start', allowing them to patch the rest of the video from other community clients or more distant nodes if necessary.

4. SYSTEM ASSESSMENT AND EVALUATION

The evaluation focuses on the central aspects reflecting the specific issues determining Corelli's behaviour. A number of requirements have been identified in Section 2, focussing on the need for responsive mechanisms to react efficiently to variations in demand, content popularity and churn. These are critical for performance in potentially divergent communities. To this end, this section firstly investigates Corelli's adaptive resource utilisation and resilience mechanisms to highlight how variations in demand and churn are managed. Following this, an overhead evaluation is provided to qualify Corelli's behaviour against its resource requirements.

Table 2. Default Simulation Configuration

Parameter	Value
Popularity Distribution	Zipf-Mandelbrot (α=0.199)
Request Distribution	Poisson (λ=3 req/min)
Object Number	7036
Client Number	160
Number of Potential Caching Peers	16 (10%)
Client Churn Distribution	Weibull (k=0.38, λ=42.4)
Bootstrap Cache Size	8 peers
Caching Peer Storage	1GB per peer
Caching Peer Bandwidth	128KBps per peer
Client Peer Bandwidth	64KBps
Video encoding	MPEG4 (426*240, 450kpbs)
Prefix Caching Size	32MB (~10 min at 450kpbs)
Caching Point	3

To evaluate these areas a discrete event based simulator has been developed in Java based on communities constructed from clients and caching peers. A Poisson distribution is used to model request rates whilst a Zipf-Mandelbrot distribution is used to model content popularity [28][29]. Using these distributions, clients then request content. If it is not cached and no community replicas exist, the client fetches the content from outside of the community. The models and simulations are configured using measurements taken from three real-world systems: China Telecom VoD [38], BitTorrent [33] and 4oD [1]. These studies provide detailed information about content and user behaviour. Table 2

provides an overview of the derived default configuration used in experiments. Through simulation, a free VoD scenario, based on [38], has been devised in which origin servers provide video content to clients. Communities of these clients then autonomously build Corelli infrastructures to augment the origin servers.

4.1 Cache Adaptability

A major requirement of Corelli is that it can adapt to changes in demand; this is necessary to ensure sufficient resources are available to serve the community. Cache adaptation consists of the addition (or removal) of peers from the distributed cache. This occurs when it is desirable to increase/reduce resources; the resources monitored are *storage* and *bandwidth*. Figure 3 shows the expansion of three independent Corelli caches (indicated by crosses). To study how Corelli behaves in different environments, each cache receives requests using a different Zipf skew based on the average, maximum and minimum skew measurements taken from [38]. This highlights how caches observing different request trends behave. Each cache is bootstrapped from a single peer and expansions are performed solely on storage monitoring to investigate the performance of storage trend analysis. Expansions first occur approximately four hours into the caches' lifecycles, with caches *A* and *B* expanding first. This is due to the fact that they receive a more skewed range of content requests than cache *C*; the trend analysis algorithms therefore predict a greater utility in expansion for these two caches. Further, cache *A* can be seen to consistently expand before the others. Both cache *A* and *B* expand to 6 peers but cache *A* commits these resource sooner to take advantage of the immediately higher hit rates possible. Cache *C* can be seen to expand the least (4) due to its more uniform distribution.

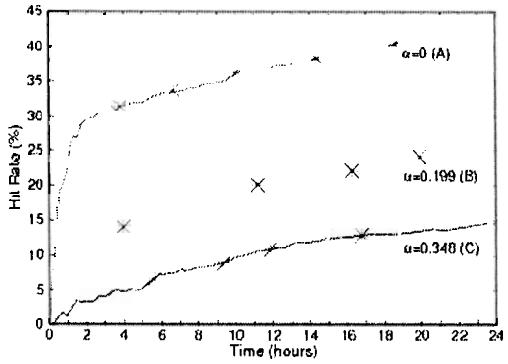

Figure 3. Hit Rate during Expansion

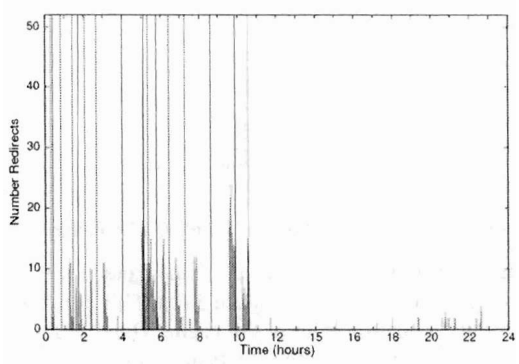

Figure 4. Client Redirection during Expansion

Adaptation is also performed to increase bandwidth resources; this occurs after complete bandwidth saturation is observed over three contiguous requests. This prevents expensive expansion procedures being undertaken during very short spikes in demand. To evaluate this process bandwidth monitoring is enabled and expansions studied. Figure 4 shows the number of redirections during the simulation caused by bandwidth saturation; this is the process of Corelli redirecting client requests to other clients in the community possessing a replica. Vertical lines represent points at which the cache expands; it can be seen that these coincide with increased levels of redirection. Subsequently, the number of redirections rapidly decreases after expansion, indicating that the expansion has adequately handled the demand. For example, the cache can be seen to receive an extended spike in demand at the 5th hour; this, however, is handled by two expansions shown by the downwards slope in redirections. Larger peaks occur further into the cache's lifecycle because, as time increases, larger amounts of content are replicated, leading to higher utilisation levels. By the 11th hour, however, sufficient resources exist in the cache (16 peers) to handle demand resulting in few subsequent redirects. The increased level of expansion based on bandwidth monitoring clearly shows that bandwidth is a more valuable commodity than storage. The demand, however, is easily handled through Corelli's adaptive algorithms.

4.2 Cache Resilience

Cache adaptation effectively addresses the dynamism observed in demand. However, a problem endemic to peer-to-peer networks is churn. The effect of churn is considerably reduced through the burn-in requirements of becoming a caching peer but it is obvious that churn still occurs in the system. It is therefore important to take this into account to ensure the resilience of deploying Corelli. To evaluate Corelli's ability to handle churn a standard configuration simulation is performed with the introduction of varying levels of churn in the caching peers. Churn is also modelled in the clients using a Weibull distribution [33]. This allows the resilience of Corelli's storage and replication policies to be investigated in the context of real-world deployment.

The effect that churn has in the system is largely dictated by two factors. The first is the replication level; a community with a high degree of passive replication is able to handle caching peer churn elegantly as requests can be easily redirected to clients. The second factor is the request rate; if churn occurs during periods of high demand the effect will be far more noticeable. This is because the clients may not be capable of handling the subsequent redirections. Figure 5 (a) shows the number of chunk request rejections attributable to the failure of a single cache at the 1st and 12th hour. It can be seen that churn early on in the cache's lifecycle has a limited effect with only a very small increase in rejections. Churn at the 12th hour, however, has a far more significant impact. This is because by this point the cache has stored a much larger number of popular objects. Therefore, the cache is serving a greater number of requests during the failure. It is also identifiable that rejections occur in two peaks; the first peak occurs due to the failure of the caching peer. This, however, is subsided through the use of redirection; the second peak occurs once the clients' resources have been saturated. This peak is dealt with by two mechanisms: firstly, the chunks from the failed caching peer are replicated onto a new caching peer and secondly, the level of passive replication is increased by the peers already downloading chunks.

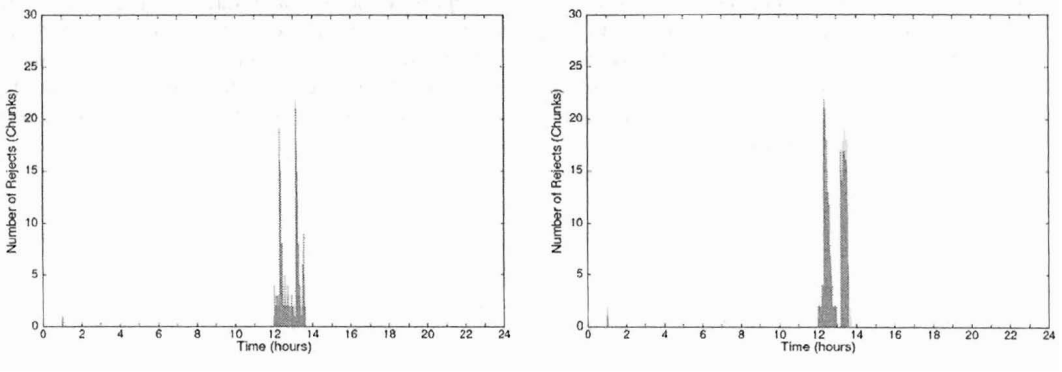

Figure 5. *(a)* Reject Rate for Single Cache Failure *(b)* Reject Rate for Treble Cache Failure

To investigate how well Corelli deals with higher levels of churn, the same experiment is repeated but removing three caches through churn, shown in Figure 5 (b). This constitutes a significant degree of failure, representing a 38% (hour 1) and 21% (hour 12) churn level in the cache. Once again, failure early on in the cache's lifecycle has a limited effect on the rejection rate. However, the effects are more noticeable at the 12th hour. The failure of a single caching peer at this point only results in two short peaks. However, the failure of three caching peers results in a more extended period of failures (represented by the width of the bars). Despite this, in both scenarios failure is quickly dealt with, showing that the effects of churn are effectively managed. Further, the overall effect of churn is minimal due to client redirection and the ability to quickly replicate popular chunks onto replacement caching peers. In all scenarios, over 95% of chunk requests can be serviced. This is achieved even with churn levels of almost 40%. This therefore shows that Corelli can maintain a high level of service, even, during significant degrees of churn. This flexibility is achieved through the process of client redirection and the ability to quickly replicate popular chunks onto replacement caching peers.

4.3 Overhead

Corelli has two forms of overhead: management and replication. *Management overhead* consists of control messages passed between caching peers whilst *replication overhead* represents the necessity to actively replicate content onto the cache. This is in contrast to traditional caching [36] that stores content passively as it passes through an ingress point.

To measure the management overhead of running a distributed cache, simulations have been performed and the communications monitored between role components. These have shown the management overhead to be minimal, with all scenarios generating under a megabyte per hour. This is because the management overhead is restricted to a subset of peers rather than fully decentralised solutions e.g. [17]. We therefore consider this overhead to be negligible.

A more important concern is the replication overhead generated. Figure 6 shows the number of chunks transferred over a 24 hour period based on the *caching point* used. The caching point represents the number of requests a cache must receive for an object before choosing to replicate it. It can be observed that a low caching point results in more chunks being transferred into the cache than provided from it. This means that Corelli will increase ingress bandwidth utilisation. This occurs because a low caching point results in more objects being eligible for replication. Importantly, however, as the caching point increases (>1), the number of chunks uploaded exceeds the number of chunks downloaded. This

indicates a high chunk reuse rate, meaning that any object replicated in the cache is likely to be subsequently accessed a high number of times. For example, a caching point of three, results in 76% of content transactions being classified as uploading content to clients rather than downloading into the cache. The overhead for caching peers during this process is also controlled; in this experiment, on average, only 22MB are uploaded onto each caching peer per hour. A further average of 34MB is also uploaded onto each caching peers per hour to address the redistribution of chunks during expansion and churn management. This, however, is necessary to ensure resilience and high bandwidth accessibility in the community. Further, redistribution is an internal process that occurs between high-power peers without the utilisation of egress points. Importantly, these experiments show that Corelli reduces the egress utilisation for popular chunks by over three quarters. This results in extremely low latency access to content and significantly reduced congestion in the egress links.

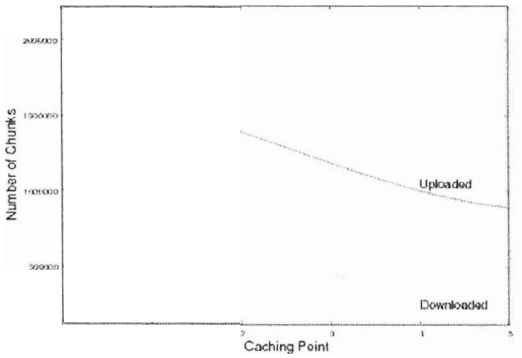

Figure 6. Chunk Upload:Download Ratio in Cache

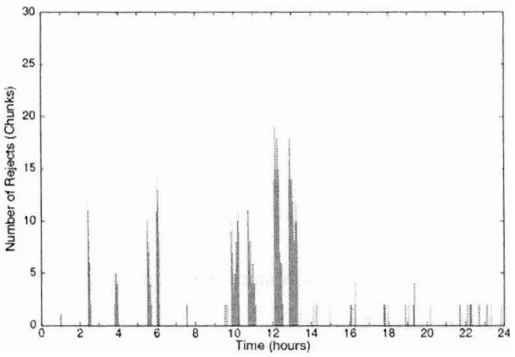

Figure 7. The Chunk Redirection Rate

Client peers in the community are also utilised in Corelli to improve performance and resilience, allowing such things as flash crowds and churn to be elegantly handled without significant deterioration in the quality of service. This, however, clearly creates an overhead for client peers; Figure 7 shows the redirection rate for a community with the failure of three caching peers after the first hour. It can be seen that significant levels of client redirection occur at various points in the cache lifecycle. These, however, are relatively short lived as the process of cache expansion quickly alleviates the client loads. For example, between the 10^{th} and 11^{th} hour, there are a number of flash crowds; during this period, the client peers are heavily relied on. Despite this, overall, only 2.87% of chunk requests are obtained through the clients, with the maximum number of clients utilised at any given time being 11.87%. This means that, the overhead required by operating as a client in Corelli is diminutive when compared to the utility gained through cooperation.

5. PERFORMANCE EVALUATION

The previous section has shown Corelli to be adaptive, resilient and capable of handling variations in demand. However, a further requirement of Corelli is that it provides sufficient performance to warrant the overhead previously detailed. To measure the performance of Corelli, simulations have been performed and various factors measured.

5.1 Hit Rate Analysis

To measure performance the hit rate is used; this represents the percentage of content requests that can be serviced by the Corelli cache. These measurements exclude the use of redirection so to solely measure the hit rates achieved by the cache. Figure 8 shows the hit rate when configured with various caching points. A low caching point increases the hit rate but also increases content turnover, whilst higher caching points reduce hit rates but also reduce content oscillation. Content oscillation is the removal then reintroduction of content in the cache. An optimal removal rate is zero, which indicates the caching algorithm executed over the period without the need to remove content. As can be seen, a low caching point (1) achieves an effective hit rate (~40%) whilst higher caching points inhibit caching and therefore lower the hit rate (e.g. 34% at point two). To ensure optimality, a cache adaptively defines its own caching point; if a cache's resource requirements cannot be satisfied through expansion, it increases its caching points incrementally until resource utilisation is manageable. If, however, there are unused resources the caching point is lowered to improve hit rates.

Alongside the hit rates achieved directly through the cache, Corelli also utilises replicas possessed by the client peers. Therefore, if an item of content is replicated in the community that does not reside on the cache, then requests are

redirected there. With a caching point of three, the hit rate is raised to 48% by utilising redirection. Similarly, ~50% is achieved using any caching point; this is because the same items of content exist in the community regardless of Corelli. Therefore, even if no items existed in the cache, the community hit rate would still be ~50%. Corelli, however, significantly aids in the distribution of the content. Without Corelli operating, only 24% of chunk requests can be handled, however, when a Corelli cache is enabled in the community this rises to 95%. Therefore, to successfully access content a non Corelli enabled community would require over 70% more external requests for popular chunks.

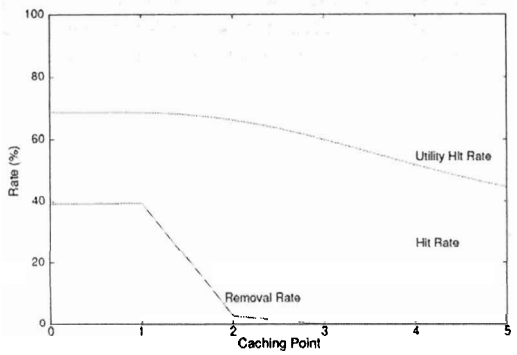

Figure 8. Cache Performance based on Caching Points

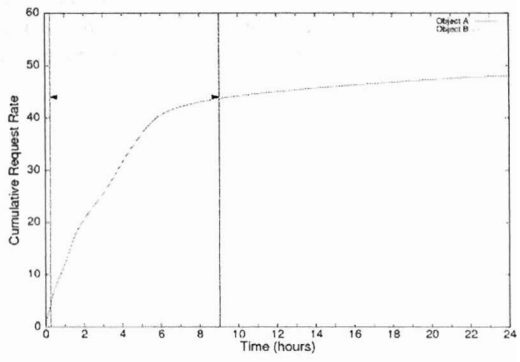

Figure 9. Cumulative Request Rates for Popular Content

5.2 Utility Hit Rate Analysis

There are a number of differences between Corelli and conventional caching; these mainly centre on its use of active replication and the resource limited nature of the peers. This means that the traditional hit rate does not offer an adequate evaluative metric for system performance. This is because the hit rate is increased by replicating an unpopular object that is requested only once; this creates an overhead that does not benefit the community. To model this, a new measurement is proposed: the *utility hit rate*. This shows the percentage of serviceable requests when only considering requests for content that is accessed more than twice. A high utility hit rate therefore indicates that the cache is making well informed decisions that result in the objects that it replicates being accessed many subsequent times. Figure 8 shows the utility hit rate for a variety of caching points. It can be seen that highly effective utility hit rates are achieved in the system. The default caching point, based on performance and overhead experiments, is three; this achieves a ~60% utility hit rate. Even higher rates are also possible with ~70% being achieved with a caching point of two; this, however, results in higher resource requirements placed on the caching peers. The utility hit rate therefore shows that Corelli can service a significant percentage of requests whilst effectively ignoring requests for low popularity items.

5.3 Demand Responsiveness Analysis

One important observation is the achievement of an optimal removal rate (0%) when the caching point exceeds two. This occurs because the cache's storage resources are not exceeded due to the frugal caching policy i.e. only replicating extremely popular content. Therefore, due to bandwidth expansion, more storage capacity than required is provisioned. This, however, does not take into account new content introduction; this can occur at regular intervals such as in [1] or in periodic clusters [38]. To evaluate how Corelli deals with new object introduction, a 24 hour simulation is performed with 30 new objects being introduced after 6 hours; these objects are placed with a 0-29 popularity ranking (i.e. Top 30). This is based on measurements taken from the China Telecom VoD system. Figure 9 shows the cumulative request rates for two objects; the first (object *A*) is originally at popularity position 0. However, after the new objects are introduced, object *A* is reduced to popularity position 30. Figure 9 also shows the cumulative request rate for a new object (object *B*) at popularity position 0. The vertical lines represent the period of time in which Corelli maintains object *A* as replicated and ineligible for removal; it can be seen that Corelli responds rapidly to the new content. Both objects *A* and *B* are replicated very early in their lifecycles (after three requests) allowing Corelli to serve the community from the beginning of the popularity. After the introduction of object *B*, the number of requests for object *A* severely drops; one and a half hours after the insertion of objet *B*, object *A* is given priority for removal.

To investigate the effect on performance, the hit rates during object introduction are also monitored; this period sees relatively little change in the hit rate (under 1%). This shows that Corelli's responsive algorithms can effectively monitor changes in request trends. To model an extreme case, simulation is also performed with the introduction of 100 highly

popular objects after 6 hours; even with this high level of content churn the hit rate is only lowered by 1.14% highlighting Corelli's responsiveness to change in popularity.

6. CONCLUSION AND FUTURE WORK

This paper has proposed and evaluated a peer-to-peer replication architecture for high bandwidth, low latency content access in community networks. It exploits the heterogeneity of peer capabilities to construct independent and self managing distributed caches from clusters of high capacity nodes therefore shielding lower power nodes from the burden. These caches monitor request trends in their community, intelligently replicating popular content early in its lifecycle. This allows community nodes to gain low latency access to popular content. Importantly, this is done transparently to any given content distribution solution. This offers significant benefits for communities that do not possess the resources to deploy dedicated caches or systems that require extremely dynamic management of their cache deployment.

Corelli has been evaluated, focussing on a number of aspects, notably, system issues (adaptability, resilience), cost issues (overhead) and performance levels. Through extensive simulation, the system has been shown to offer an adaptable and resilient alternative to static replication or more traditional caching approaches. A performance evaluation has shown that effective hit rates are achieved (30-40%) whilst community hit rates are even higher (~50%). Importantly, it has also been shown that Corelli increases the accessibility of popular chunks in the community by over 70% showing that communities without Corelli would have higher egress utilisation alongside greater latency for popular content.

Further work has also been carried out, including incentive management and quality of experience (QoE) experiments, however, this has not been included due to space constraints. There are also a number of promising extensions that can be carried out; most notably, the deployment of a prototype. Further investigation is to be performed into applying the Corelli philosophy to other aspects of multimedia distribution. Also, the issue of security must be addressed; this will involve investigating how Corelli can be protected from non-cooperative and malicious peers.

7. ACKNOWLEDGEMENTS

The authors would like to thank Yehia El-khatib and Zoë Foster for their valuable contributions during the preparation of this research. This work is supported by the European Network of Excellence CONTENT (FP6-IST- 038423).

REFERENCES

[1] 4oD – Channel 4's TV and Film on Demand Service. *http://www.channel4.com/4od/index.html*

[2] Annapureddy, S., Freedman, M.J., and Mazires, D. Shark: Scaling File Servers via Cooperative Caching. In Proc. USENIX/ACM Symposium on Networked Systems Design and Implementation , Boston (2005)

[3] BBC News. BBC and ISPs clash over iPlayer. *http://news.bbc.co.uk/1/hi/technology/7336940.stm*

[4] BitTorrent Specification. *http://www.bittorrent.org/beps/bep_0003.html.*

[5] Bommaiah, E., Guo, K., Hofmann, M., and Paul, S. Design and Implementation of a Caching System for Streaming Media over the Internet. In Proc. IEEE Real Time Technology and Applications Symposium (2000)

[6] Brampton, A., MacQuire, A., Rai, I. A., Race, N. J. P., Mathy, L., and Fry, M. Characterising User Interactivity for Sports Video-On-Demand. In Proc. ACM NOSSDAV, Urbana, Illinois (2007)

[7] Bury, S. and Race, N.J.P. Towards Resilient Community Wireless Mesh Networks. In Proc. Intl. Conference on Autonomous Infrastructure, Management and Security, Bremen, Germany (2008)

[8] Champaign-Urbana Community Wireless Network CUWiN. *http://cuwireless.net/*

[9] Cherkasova, L. and Gupta, M. Characterizing locality, evolution, and life span of accesses in enterprise media server workloads. In Proc. ACM NOSSDAV, Miami, FL (2002)

[10] Darlagiannis, V. Overlay Network Mechanisms for Peer-to-Peer Systems. PhD Dissertation. Technical University of Darmstadt (2005)

[11] Deshpande, M., Amit, A., Chang, M., Venkatasubramanian, N., and Mehrotra, S. Flashback: A Peer-to-Peer Web Server for Flash Crowds. In Proc. Intl. Conference on Distributed Computing Systems, Toronto, Canada (2007)

[12] Fan, L., Cao, P., Almeida, J., and Broder, A. Z. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. IEEE/ACM Transactions on Networking. 8, 3 June (2000)

[13] Freedman, M.J, Freudenthal, E., and Mazières, D. Democratizing content publication with Coral. In Proc. USENIX Network System Design and Implementation, San Francisco, CA (2004)
[14] Grace, P., Blair, G.S., and Samuel, S. ReMMoC: A Reflective Middleware to Support Mobile Client Interoperability. Proc. Intl. Symposium of Distributed Objects and Applications, Catania, Italy (2003)
[15] Guo, L, Chen, S., Ren, S. Chen, X., and Jiang, S. PROP: A Scalable and Reliable P2P Assisted Proxy Streaming System. In Proc. International Conference on Distributed Computing Systems (2004)
[16] Ip, A.T.S, Liu, J., and Lui, J.C.S. COPACC: A Cooperative Proxy-Client Caching System for On-Demand Media Streaming. In Proc. Intl. IFIP Networking Conference, Waterloo, Canada (2005)
[17] Iyer, S., Rowstron, A., and Druschel, P. Squirrel: A Decentralized, P2P Web Cache. In Proc. Annual ACM Symposium on Principles of Distributed Computing , Monterey, CA (2002)
[18] Kangasharju, J.; Ross, K.W.; Turner, D.A., Optimizing File Availability in Peer-to-Peer Content Distribution. In Proc. IEEE INFOCOM, Anchorage , Alaska.(2007)
[19] Korupolu, M. R. and Dahlin, M. Coordinated Placement and Replacement for Large-Scale Distributed Caches. IEEE Transactions on Knowledge. and Data Eng. 14, 6 Nov (2002)
[20] Laoutaris, N., Telelis, O., and Zissimopoulos, V. Distributed Selfish Replication. IEEE Transactions Parallel Distributed Systems 17, 12 Dec (2006)
[21] Leff, A., Wolf, J. L., and Yu, P. S. 1993. Replication Algorithms in a Remote Caching Architecture. IEEE Trans. Parallel Distribution. Syst. 4, 11 Nov (1993)
[22] Padmanabhan, V.N., and Sripanidkulchai, K. The Case for Cooperative Networking. In Proc. Intl. Peer To Peer Systems Workshop, Cambridge, MA (2002)
[23] Patel, A., and Gupta, I. Overhaul: Extending HTTP to Combat Flash Crowds. In Proc. of Web Content Caching and Distribution, Beijing, China (2004)
[24] Patterson, D. A., Gibson, G., and Katz, R. H. A Case for Redundant Arrays of Inexpensive Disks (RAID). SIGMOD Rec. 17, 3 p109-116 (1988)
[25] Preuss, S. JESA Service Discovery Protocol. In Proc. IFIP Networking, Pisa, Italy (2002)
[26] Rowstron, A., and Druschel, P. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In Proc. ACM/IFIP Middleware, Heidelberg, Germany (2001)
[27] Rowstron, A., and Druschel, P. Storage Management and Caching in PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility. In Proc. ACM Symposium on Operating Systems Principles (2001)
[28] Saleh, O., and Hefeeda, M. Modelling and Caching of P2P Traffic. In Proc. of Intl. Conference on Network Protocols, Santa Barbara, CA, (2006)
[29] Saroiu, S., Gummadi, P.K., and Gribble, S.D. A Measurement Study of Peer-to-Peer File Sharing Systems. Technical Report # UW-CSE-01-06-02, University of Washington (2002)
[30] Sen, S., Rexford, J., and Towsley, D. Proxy Prefix Caching for Multimedia Streams. In Proc. IEEE INFOCOM, New York, NY (1999)
[31] Skevik, K. The SPP Architecture: A System for Interactive Video Streaming. PhD Dissertation, Uni. of Oslo. (2007)
[32] Stading, T., Maniatis, P., and Baker, M. P2P Caching Schemes to Address Flash Crowds. In Revised Papers from the 1st Intl. Workshop on P2P Systems (2002)
[33] Stutzbach, D. and Rejaie, R. Understanding Churn in Peer-to-Peer Networks. In Proc. ACM SIGCOMM Conference on Internet Measurement, Rio de Janeriro, Brazil (2006)
[34] Tran, D., Hua, K. and Sheu, S. 2003. Zigzag: An Efficient P2P Scheme for Media Streaming. In Proc. IEEE INFOCOM, San Francisco, CA (2003)
[35] Uppalapati, S. and Tosun, A.S. Partial Video Replication for Peer-to-Peer Streaming. In Proc. Intl. Conference on Management of Multimedia Networks and Services, Barcelona, Spain (2005)
[36] Wang, J. A Survey of Web Caching Schemes for the Internet. Technical Report. UMI Order Number: TR99-1747., Cornell University (1999)
[37] Wong, B., Slivkins, A., and Sirer, E. G. Meridian: a Lightweight Network Location Service without Virtual Coordinates. SIGCOMM Computing. Communications. Rev. 35, 4 (2005)
[38] Yu, H., Zheng, D., Zhao, B. Y., and Zheng, W. Understanding User Behaviour in Large-Scale Video-on-Demand Systems. In Proc. ACM Sigops/Eurosys European Conference on Computer Systems (2006)
[39] Zhang, X., Liu, J., Li, B., and Yum, T.S. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In Proc. IEEE INFOCOM, Miama, FL (2005)
[40] Zhong, M., Shen, K., and Seiferas, J. Replication Degree Customization for High Availability. In SIGOPS Operating Systems Review 42, 4 April (2008)