International Journal of Enhanced Research in Science Technology & Engineering, ISSN: 2319-7463
Vol. 4 Issue 2, Feb.-2015, pp: (159-179), Impact Factor: 1.252, Available online at: www.erpublications.com

Distributed Cache

Nipun Bansal[1], Megha Bansal[2], Prateek Roopra[3]
[13]SGGSCC, University of Delhi, New Delhi, India
[2]Gargi College, University of Delhi, New Delhi, India

Abstract: Replication and caching strategies are increasingly being used to improve performance and reduce delays in distributed environments. The benefits of using caches for reducing traffic in backbone trunk links and for improving web access times are well-known. A query can be answered more quickly by accessing a cached copy than making a database round trip. This paper includes an introduction of distributed cache and an adaptive algorithm for distributed caching based on the idea of autonomous proxy caches without the usage of a central coordinator or broadcasting protocol and its applications in various areas such as cellular mobile networks, web cache, distributed file system and web services, databases. In this paper we analyzed the performance of both hierarchical and distributed caching. Our main performance measure is the expected latency to retrieve a Web document. It describes deficiencies in an existing hierarchical caching scheme for document replication and presents an alternative approach that more widely distributes the load across multiple servers, and provides additional resource discovery features.

Keywords: Adaptive Distributed Cache, Cache Invalidation, Concurrency Control, Hierarchical Distributed Cache, MemCache, Web Cache,

I. Introduction

A distributed cache is a clustered, fault-tolerant cache that has linear scalability. It is scalable because of the architecture it employs. Data is partitioned among all the machines of the cluster. For fault-tolerance, distributed caches can be configured to keep each piece of data on one or more unique machines within a cluster. For application data, it keeps a copy of a subset of the data in the database. It distributes its work across multiple servers but still gives you a logical view of a single cache. This is meant to be a temporary store, which might mean hours, days or weeks. In a lot of situations, the data being used in an application does not need to be stored permanently.

Distributed environments involve databases and applications, data are often stored on a central database server. When sending a query or update request directly to the database server, an application on one of the application servers in the cluster sends the query (SQL) over the network. On receipt of the request from the database server, the application instantiates a new object and populates it with the new data. Each such data exchange introduces latency and increases CPU and network usage. Thus applications can easily flood database servers with queries and transactions in those domains that require continuous intensive data exchanges between the database and applications.

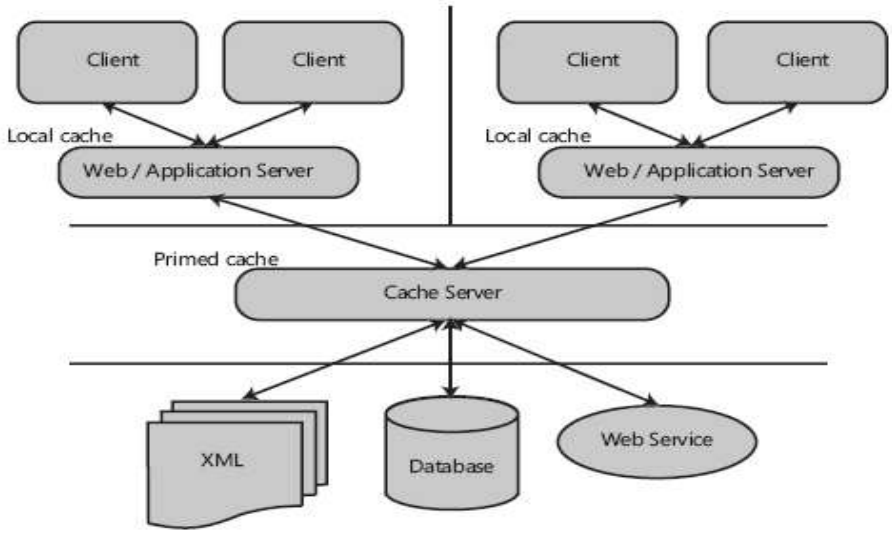

Figure 1: Distributed Cache Architecture

A. Cache Invalidation And Consistency Model

A popular approach to overcoming these problems is to cache data from the database on the application side. One challenge of this approach is to maintain coherency of data in a cluster, thus ensuring data integrity even for cached data. To overcome this problem, an invalidation policy can be used to inform caches about updates and to cause the caches to update the invalidated data. Invalidation policies can help ensure data integrity. In [1] paper the author report the results of experimental study of invalidation protocols in systems that use distributed data caching.

1) Cache Consistency: There are two consistency models

Weak consistency model: Weak consistency model consist of TTL-based schemes and client polling. In TTL-based schemes, each object retrieved from the database gets an associated time to live (TTL). When the object is referenced, its TTL is checked for validity. If the check is successful, the object is retrieved; otherwise a new value is requested from the server. They compared and contrasted two kinds of TTL-based schemes: fixed TTL, where an object TTL is set to a fixed value whenever an object is retrieved from the server, and adaptive TTL, where the TTL is a function of the frequency of object access.

Strong consistency model: For strong consistency model they consider polling each time and both client-side and server-side invalidation. In polling each time, client servers send a validity request to the origin server each time a request for an object hits the cache. In server-side invalidation, the origin (i.e., database) server keeps track of all the clients that have cached an object. When the object is modified, the cache server informs the origin server, which in turn sends invalidation messages to all other clients; a write is considered complete when the invalidation messages have reached all the relevant clients.

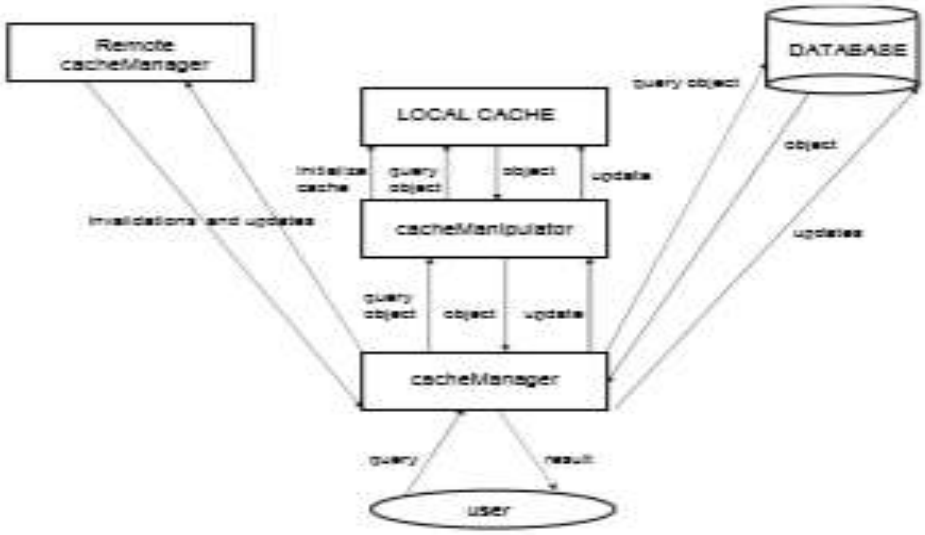

Figure 2: Layout of TestBed

2) Experimental Set: The experimental set up is shown in Figure 2

3) Evaluation:

In the experiments, the author ran all transactions in simultaneous batches and used the number of transactions as the measure of the load run at the servers. They consider three values of server load: low (between 1 and 10 transactions), medium (10 to 100 transactions), and high (greater than 100 transactions). They also used latency, that is, end-to-end delay in milliseconds required to complete a bulk transaction by a cache server. These experiments shows that the scheme which always proves to be most costly is polling each time, which is a pull-based scheme. It performs 150 times worse than the other schemes in terms of average times. In addition, results suggest that push-based schemes tend to perform better than pull-based invalidation schemes. Thus it is conclude that server-side invalidation is highly recommended for small-size networks as it is guaranteed to never return a stale object.

B. Distributed Cache

One approach which address the problem of servicing the demand of growing Internet is the replication of information sources within an organization. This can greatly reduce the number of redundant fetches over heavily utilized links and can also help to alleviate the load on the servers at popular sites. Replication is used to improve the reliability and performance of data access. Different approaches are used to implement replication than are used in more structured

distributed systems: a) Mirroring the periodic replication of a collection of less belonging to a particular Internet site or archive is often called mirroring. b) Caching One alternative to mirroring is caching. In this scheme, when a file is downloaded by a user it is stored locally so that subsequent requests may retrieve the cached copy. This form of replication is referred to as Internet caching.

Hierarchical Cache

The goals of the hierarchical cache are to distribute load away from server hot spots and to reduce access latency. The server which is at the upper most level of the hierarchy is termed the root cache, while those nodes at the lowest levels of the hierarchy are termed leaf caches. The node which holds the original copy of the document is called the primary site. In this approach, www clients are configured to request a document from a leaf cache using a standard HTTP proxy protocol. If the request cannot be resolved from its local set of documents, the cache then queries its parents and siblings using a datagram oriented protocol known as the Internet Cache Protocol (ICP).

Advantages

1) It can reduce wide area bandwidth demand by reducing the redundant transfer of information at successive levels in the network.
2) The implementation can also help to further reduce the load on server 'hot spots' in the network by decreasing the number of accesses to such nodes.
3) The root node is responsible for processing all requests which are not resolved by the other caches in the network, therefore it can experience high load when the number of requests from these other caches is large.
Disadvantages
1) The root node and other upper level caches, the storage space required to hold this cache is thus likely to be quite large.
2) As the number of users and file sizes increase, the efficiency of the hierarchical approach will degrade.
3) This mechanism creates management problems because each node must maintain location information about all its siblings to enable queries to be directed to them.

Distributed Cache

One approach which addresses the above issues is to avoid the use of servers at the root and upper levels of the hierarchy to resolve requests and store cached documents. Instead, only the leaf caches would be responsible for retrieving objects, while the upper level caches would be used to maintain information about the contents of these caches.

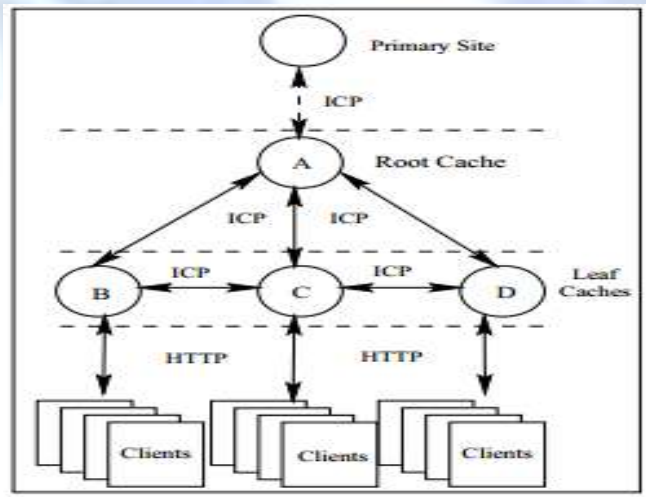

Figure 3: Hierarchical Cache

In this case, neither nodes A, B or C are caching objects. They are instead used only to propagate cache information. For example, if node F requires an object, it queries node B (step 1). Here node B does not know where a copy of the object can be found. However, instead of returning a 'miss' immediately to F, it propagates the query to A (step 2). Node A is also unable to locate the object returns a miss to B which then propagates this information back to F. This recursive querying approach allows the entire cache contents to be searched very quickly so that node F can correctly determine that there is no cached copy of the requested object and thus retrieve it from its primary site (step 3). When a miss is encounteredby a leaf cache and it resolves the request from its primary site, a mechanism is required to indicate to the upper level nodes in the hierarchy that the document has been cached. This procedure is known as an 'advertisement.(part 4). Node F sends an advertisement message to its parent (B). The advertisement is then propagated

recursively to the root node. If a child of node B then requests the object then B will have information about where to find it. If a child of node C requests an object then by recursively querying, the root node A will indicate that a copy may be found at F (step 5). Rather than returning the actual object from its cache, the upper level server will return a 'pointer' to the cached object consisting of a Uniform Resource Locator (URL) and information about the size and modification time of the object. In the example, when E retrieves the object it may also advertise a pointer to C and A (step 6).

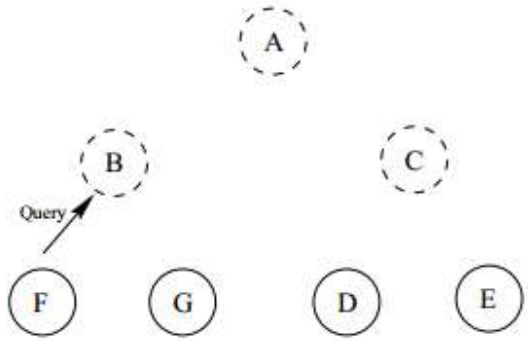

Figure 4: F queries B to see if it has a copy of the object

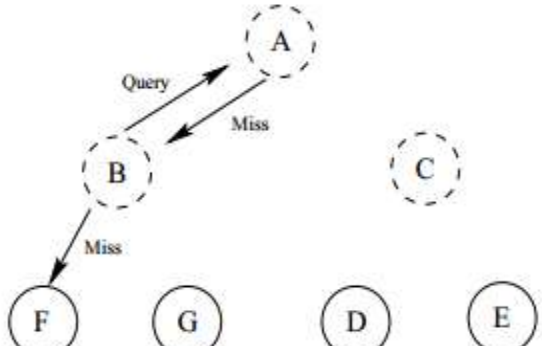

Figure 5: B does not have an entry for the object and forwards the request to A.A does not have an entry either. The miss is returned back to F

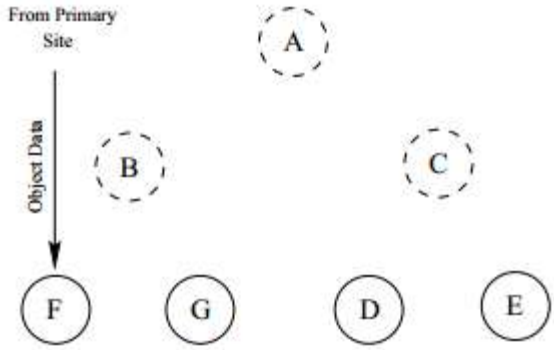

Figure 6: F retrieves the object from its primary site

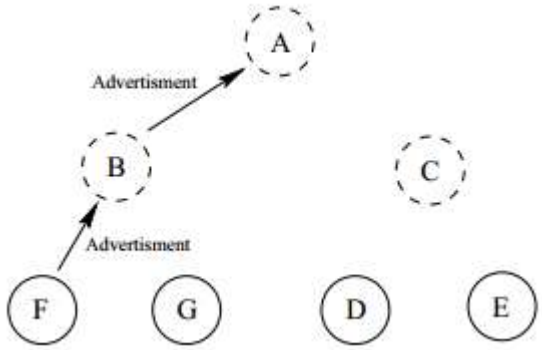

Figure 7: F advertises its cache recursively up the hierarchy

Advantages

1) Removes disadvantages of hierarchical caching.
2) The advertisement scheme could be extended to allow the system to integrate the propagation of information about mirror sites.

Disadvantages

1) It increases the load on leaf servers.
Thus distributed cache has various advantages over hierarchical cache. [16] gives the numerical comparison of distributed and hierarchical cache. Also it evaluates performance, connection time, transmission time, total latency, bandwidth and cache load, for which two models have been used i.e Network Model and Document Model.

Network Model

Full O-ary tree
O - The nodal out degree of the tree H is the number of network links between the root node of a national network and the root node of a regional network. H is also the number of links between the root node of a regional network and the root node of an institutional network. z is the number of links between a origin server and root node (i.e., the international path). Let l be the level of the tree.

Document Model

O-ary trees are good models. Modifying the height or the number of tiers of the tree can easily model other networks.The model assumes homogeneous client communities. Heterogeneous client communities can be easily modeled. Simulations results in this paper should be considered as relative results.

Total latency,T to fetch a document is divided into two parts: connection and transmission time. Connection time (T_c) is the time since the document is requested by client and first byte of data is received. Transmission time (T_t) is the time to transmit the document. Total latency=Connection Time + Transmission Time $E[T]= E[T_c]+E[T_t]$
Total latency is used to compare hierarchical and distributed cache with respect to their connection and transmission time and the performance has been shown graphically below. Connection Time depends on the number of network links from the client to the cache. Where, H: link between root nodes, z:links between server and root nodes, l:level of the tree

Li: link that a request to document i travels d:the per hop propagation-delay $E[T_c^h]$,$E[T_c^d]$ are the connection time for hierarchical and distributed caching. $E[T_t^h]$,$E[T_t^d]$ are the transmission time for hierarchical and distributed caching.

$$E[T_c^h] = 4d \sum t \in \{0, H, 2H, 2H + z\} P(L = l)(l + 1)$$

$$E[T_c^d] = 4d \sum_{t=0}^{2} H\ P(L = l)(2l + 1) + 4d\ P(L = 2H + z)(2H + z + 1)$$

$$E[T_t^h] = \sum t \in \{0, H, 2H, 2H + z\} E[T_t^h \mid L = l]\ P(L = l)$$

$$E[T_t^d] = \sum_{t=0}^{2H+z} E[T_t^d \mid L = l]\ P(L = l)$$

For determining these 2 terms $E[T_c^h \mid L = l]\ E[T_c^d \mid L = l]$,request arrival rate for network level is determined first The aggregate request rate generated by hierarchical caching between the level l and l+1 of network tree is given by: β_h^t is the request arrival rate for hierarchical caching

β_d^t is the request arrival rate for distributed caching, β_d^t is the request arrival rate for distributed caching
$\beta_t^d = O^t\ \beta_t\ ((1\text{-}hit_t)+(hit_N - hit_t))$ for , $0< l <2H$and by, $O^{2H}\ \beta_t\ (1\text{-}hit_N)$ for, $2H < l < (2H+z)$

A hybrid caching scheme can combine the advantages of hierarchical and distributed caching, thus reducing connection time and transmission time. Cache clustering is a natural way to scale as traffic increases. In [8] the idea is to replace a single cache, when it is no longer able to handle all the traffic, with a cluster of caches. However, cache clustering introduces a new request-routing problem. The two most popular approaches to address this problem are:

1. The cluster is arranged in a tightly-coupled manner
2. The cluster is arranged in a loosely-coupled manner - Each proxy server can serve each web item, and therefore (a) this approach can guarantee a much better load balancing; (b) scaling, by and (c) this approach is less sensitive to proxy failures.

Generally, two types of directors are used: a Layer-5 (Application Layer) director and a Layer-4(Transport Layer) director. The main approach for implementing a Layer-5 director is using another proxy server. Such a proxy terminates the TCP connection with the client, views and analyzes the client request, determines the best proxy machine within the cluster that can serve the request, and forwards the request to this proxy over another TCP connection. In contrast, a Layer-4 director determines the target machine upon receiving the client TCP SYN segment, and therefore without reading the request data. It determines the most available cache proxy, and assigns the new connection to this machine. This usually involves changing destination IP address of the connections packets to the internal IP address of that machine through a process known as NAT.

Cache Pruning Algorithm
A cache pruning algorithm is periodically invoked in order to remove web items from a cache until the cache free space reaches some predetermined threshold. Cache pruning logic for traditional CPU or I/O subsystems usually makes a decision based on one of the following two parameters:

1) Least Recently Used
2) Least Frequently Used
3)The time it takes to retrieve each item from its original server.
4) Whether the web item exists or not exists in another proxy within the same cluster.
Cache Pruning Problem: Caching gain can be measured as an Object-Hit-Ratio (ratio of the number of requests served from the cache to the total number of requests) or as Byte-Hit-Ratio (ratio of the number of bytes served from the cache to the total number of served bytes).

The Last Copy Concept
Assume a mechanism that guarantees that exactly one copy of every web item in the cluster is marked as Last-Copy. When the proxy runs its single-proxy pruning algorithm, it assigns a lower removal priority to each local item marked as Last-Copy. The purpose is to guarantee that even if an item is pruned from the cache of the other proxies in the cluster, its Last-Copy remains. Each proxy maintains two groups of items, a group of Last-Copy items and a group of Not-Last-Copy items.

A Distributed Intra-Cluster Scheme based on the Last-Copy Concept:

1) Only one copy of each web item in the cluster is classified as Last-Copy. An exception to this property might be if two requests for the same web item are received almost simultaneously by two different proxies when a Last-Copy for this particular web item does not exist in the cluster. In such a case, both proxies will mark their copy as Last-Copy. However later on one of these proxies will re-mark its copy as Not-Last-Copy after the copy is requested by a third proxy. If one or more copies of a web item exist in the cache, a request for this item will be served by the local cluster rather than by the original server.

2) One may consider two models for the management of a cluster with the Last-Copy attribute: 1. A centralized model, where all the cluster participants are controlled by a centralized node that guarantees properties 1-2 above. 2. A distributed model, where no centralized controller exists. In such a case the cluster proxies need to run some intra-cluster protocol in order to maintain properties 1-2 above.

ICP is a lightweight protocol, used for inter-proxy communication. In the proposed scheme, when a proxy receives a request for a web item that is not found in the local cache, it multicasts an ICP query message to the group of proxies in the cluster. Suppose that a proxy server Pi receives a user request for a web item w. The following cases are possible: 1. w exists in the local cache and it is valid. 2. w exists in the local cache but it is not valid. 3. w does not exist in the local cache. The response of Pi to the user request is as follows:

1. If w exists in the local cache and it is valid, it is sent to the requesting user.
2. If w exists in the local cache but it is not valid then: (a) If Pi (w), namely the copy of w at Pi , is marked as Last-Copy, then Pi checks the validity of Pi (w) against the original server. If Pi (w) is not fresh, Pi gets a fresh copy of w from the original server and sends it to the requesting user. The status of the fresh copy remains Last-Copy. (b) If Pi (w) is not marked as Last-Copy, then w is requested from the local proxy that maintains the Last-Copy of w. Let this proxy be Pj . Proxy Pi discovers Pj by multicasting a special ICP query message called Last-Copy-Search. The protocol then continues as follows: i. If Pj (w) is valid, then Pj sends w to Pi, and Pi sends it to the user. ii. If Pj (w) is not valid, then Pj informs Pi by unicast that the Last-Copy is not valid. In addition Pj marks its copy as not-Last-Copy. Proxy Pi requests a fresh copy to be cached from the original server, marks it as Last-Copy, and sends it to the user. iii. If no reply from a Last-Copy owner of w is received within a time- out period, Pi requests a fresh copy to be cached from the original server, marks this copy as Last-Copy, and sends it to the requesting user.

3. If w does not exist in the local cache, Pi sends a multicast Last-Copy-Search and continues as in step 2(b) above. The Last-Copy scheme increases the Object-Hit-Ratio and the Byte-Hit-Ratio substantially. The relative contribution of the scheme is only slightly affected by the cache size. Since the scheme increases the availability of less popular items, namely larger items in their trace, it is evident that for both LFU and LRU the contribution of Last-Copy to the Byte-Hit-Ratio is higher than its contribution to the Object-Hit-Ratio. Another interesting result is that without the Last-Copy scheme, the Object-Hit-Ratio and Byte-Hit-Ratio of LRU are much better than for LFU. However with the Last-Copy scheme the differences disappear and both algorithms perform very similarly.

A new architecture is introduced in [9], which treats cache as a collective resource and therefore solve some of the fundamental caching problems as described above. To give benefits of caching to the end-user in transparent manner, and to forward end-user's web requests transparently is to deploy a layer-4(L4) switch in the physical path between the machine and cache. The L4 switch can then trap any traffic destined for port 80 (web requests) and to a cache with end-users knowledge. In cooperative cache for eg. Internet cache protocol, which is used to check if the requested object reside at the peer cache, ICP requires queries be sent sequentially to a preconfigured list of peer caches and if some cache has the requested object, an HTTP GET request is sent to that cache to fetch the object. This is inefficient mechanism as every miss results in sequence of ICP messages and the messages are not sent parallely. These drawbacks avoided by maintaining summary of web objects stored locally at the peer cache and hence make local decision about which cache to go for specific web object. This avoids sequential polling of ICP known as summary cache/cache digest. Advantage- It improves hit ratio Disadvantage- Do not help in balancing load across no of caches or in reducing traffic for frequently changing web objects.

WebWave is a diffusion based protocol for hierarchical structure of caches.
Advantage- It helps load balancing between the caches and improves client response time.
Disadvantage- But it is not effective in improving hit ratio or in reducing traffic across backbone trunk links.

WCCP is another protocol used between router and set of caches mainly to provide transparent redirection of HTTP requests to the caches, also provide load balancing amongst caches. DAC3 considers set of n caches deployed in a network not as an independent caches, but rather as an aggregate pools such that web objects are stored in any of these caches. It can be retrieved by a client request.

Benefits - Hit ratio improves by 25 % to 40% .For example if object is in cache Ci and the request for the web object was generated from a client group CGj where i NOT EQUAL j and the web object is not cached in Cj, it is server from Ci, resulting in a hit. Freshness of the objects served from the caches improves by 60% for frequently changing objects because of centralised checking and update of objects. Traffic handling capacity of network improves by 25% as load for popular objects are distributed among caches. Traffic through backbone link reduces by 45% because only one cache is responsible for checking the freshness and fetching modified objects through backbone trunk link. DAC3 uses a central control station called Web Controller which monitors the traffic, identifies hot objects dynamically.

Data Flow Architecture

It consist of: Active webCache,WebCache and WebDirector Active Web Cache maintains with it list of hot objects and checks for freshness of hot objects.
WebCaches are set of machines which cache objects distributed over network.
WbDirector is an L4 switch which forward the HTTP request based on five tuple (source IP, destination IP, source port, destination port ,protocol) too WebCaches.
When a HTTP request arrives at WebDirector,it looks up a forwarding table and dispatches the request to one of the webCaches.

Control Flow Architecture
It consist of WebCaches,WebDirector,WebController,ActiveWebCache.
Each WebCache uses hot-object threshold Tho such that if no of access to an object over a period of time exceeds that threshold, it is marked hot. WebDirector distributes requests for a givent site among multiple webCaches with different probabilities. Finally WebDirector probes message to WebCaches to check for their liveness. WebController collects information from different components of the system, consolidates the information and distributes relevant information to different components.

One of the application of cache is on mobile devices: Mobile devices often rely on cellular connections to retrieve application data. Environmental factors, however, can partially or completely restrict cellular connectivity. There exists autonomic systems where distributed caching mechanisms can be used to allow mobile device networks to self-heal by storing data needed across multiple devices, but cannot be applied without a means to determine if devices are within a given range.

Main issues
When retrieving cached data, since the number of roundtrips to the database are reduced.
Data are often distributed and replicated over several caches to improve memory and CPU utilization. Since the data are stored locally in this case, local data updates on these individual caches may lead to situations where the information in the database and the cache drift out of synchronization. This may cause invalid data and results being returned. Concurrency control deals with issues involved with allowing multiple end users simultaneous access to shared entities, such as objects or data records. The goal is to come up with heuristics to choose some concurrency control mechanisms over others, depending on scenarios such as the number of data requests and the ratio of read to write requests. With distributed caching have shorter transmission time and higher bandwidth than hierarchical caching and network traffic generated by distributed scheme is better distributed uses more bandwidth in lower network layer which are less congested.

To implement the feature of concurrency control in client- server architecture is which concurrency control algorithms would perform better for maintaining cache coherency for several scenarios in distributed caching
Approach based on the non-locking model (Single thread model)

This scheme is an implementation of the remote read and remote write or single server model. The requests to the cache are buffered and sent to the origin server, which hosts the primary copy. The origin server buffers the requests and processes them in serial order. No locking is involved as there are no concurrent options. The cache server is a remote server that implements an object cache. The origin server is the server where the original stored data resides. The lock manager resides on the origin server and is responsible for assigning the locks.

1: this is code . ►this is a comment
2: a ← b
3: b ← r
4: r ← a mod b

Approaches based on the locking model read requests and write requests are synchronized by explicitly detecting and preventing conflicts between concurrent operations. With single synchronisation variable (S).

Locking with no preference:
This scheme is an implementation of primary copy locking or remote read and remote write .In this scheme, concurrency control is implemented by using mutex locks. Before a request can perform a read or write operation, it must acquire a mutex lock for that operation.
Locking for updates with write-behind cache
Implements its caching in such a way that all read-only operations occur locally, all concurrency control operations involve at most one other cluster node, and only update operations require communicating with all other cluster nodes. The scheme uses the concept of an issuer. An issuer is a node that is responsible for maintaining the consistency of the data it is caching.
An experiment conducted using testbed which contained origin server, cache server, objects, cache, and request. Object are java objects kind of integer. Request contain the actual any SQL query to fetch the result from the table.
Evaluation of Experiment:
Evaluated by measuring performance in terms of average latency experienced while completing a batch of requests. The requests were in the form of SQL queries which either read or update an object. Then batched the SQL queries together and ran them on a client server machine. Then varied the number of requests in each batch to measure the performance of the scheme under different load settings.
Specifically, for each scheme the average latency values were recorded for the low, medium and high number of requests. The ratio of read to write requests was also varied. The degree of replication of the caches was varied as well. The results of all the schemes compared and the average latency times experienced when a client server completes read and update requests in the ratio 1:1. These graphs are indicative of a low load of 10 requests, a medium load of 100

C. Adaptive Distributed Caching

Adaptive Distributed Cache is a distributed cache which adapts itself dynamically by adaptive algorithm. The Common approaches for cooperative systems encompass mostly solutions based on hierarchical cache structures or classical hashing algorithms. The hierarchical approach leads usually to multiple copies of the same object in different locations, minimizing the overall cache usage but allows good load balancing in hotspot situations. The hashing approach assigns one exact location to each object, leading to a maximum of cache usage but lacks the flexibility to load balance hotspots through multiple copies. The major objectives of a distributed cooperative proxy environment are: Object Allocation, Cache Usage, Load Balancing and Reactivity towards the Infrastructure.

The experienced request pattern on a proxy server follows the power law distribution, where a small number of most objects get requested most of the time and a high number of objects get requested very rarely. In the other extreme the small set of mostly requested objects, hot documents, is located on the same proxy, leading to a bottleneck scenario in preassigned hashing algorithm with lack of maintaining multiple objects. An ideal cooperative proxy environment maintains multiple copies of the same hot object on different locations to load balance the proxy latency. [17] looked for an adaptive distributed algorithm that was able to maximize the average hit rate and minimize the average number of hops needed to find the document.

Each autonomous proxy has to be proactive towards the ideal system state through request response evaluation and request pattern observation. [17] proposed an algorithm which came with two adaptive components, which reinforced each other in the global picture: selective caching request forwarding. Selective Caching: The adaptive proxy tries to stabilize its cache content in the currently most often requested objects. This stabilization allows neighbouring proxies to make good assumptions about the cached content.

Request Forwarding assigns an unresolved request to the most suitable proxy, which is most often returning a cache hit. The more proxies agree on the same location, the more the targeted proxy will experience a change in its request pattern and cache the highly requested objects.

The proxy stores information for each requested object in the mapping table. The table rows represent the entry for one object, while the table columns represent the attributes: Object ID, Location ID, Stability, Time Stamp, Average Time.
When an incoming request was not resolved by the locally cached data, an alternative route needs to be selected based on the information in the mapping table. The proposed algorithm simply accepts the assigned location with a probability of the stability value. If the assigned location is equal to the current proxy or if a loop got detected, the request will be forwarded directly to the origin server. A newly requested object will always be initialized with a very small stability value (0.02) leading to a high chance for a random search across multiple proxies to resolve the request.

After either a proxy or the origin server resolved the request, the package traverses the same way back to the requesting client and leaves a feedback trail in each passing proxy. Each proxy will individually decide whether to store or discard the received data before forwarding the resolved object to the next proxy on the feedback path. Before the package migrates its way back, it will be marked with the ID of the resolving proxy or the ID of the proxy which forwarded the request to the server. In case, where the request got resolved by the server and not by a proxy, the first proxy on the way

back that caches the object will overwrite the current resolver ID with its own ID, to inform the remaining proxies on the response path about its ability to resolve future requests much faster.
To allow the cache to stabilize in a set of objects, they followed the idea of selective caching. Each returned object from a forwarded request has to fulfill a certain criteria to enter the cache.
For comparison and evaluation, they used the average request time for a certain object plus the current time difference to the last request to allow object aging. For a new object to enter the cache, it has to pass the cached object with the worst request frequency in regard to the current time. If a cache hit occurs on a specific object, the object will slowly climb to the top of the sorted cache while the other objects age and move to the bottom. If a new object is able to enter the cache, the cached object with the worst request frequency will be removed; leading to stabilization in the most frequently used objects. In their experiments, they compared their algorithm to two types of hashing algorithms (Hashing with Cache Everything and Hashing with cache assigned objects only) and showed that their approach was able to outperform both of them in significant points like hit rate, number of hops and load balancing. In the first simulation, they ran all three algorithms against a typical ZIPF distribution over a set of 10000 web objects with a set of 10 proxies providing a cache space of 1000 objects. Average Hit Rate shows that the hashing approach with storage of only assigned objects (HNC) outperforms the hit rate of the hashing approach with storage of all objects (HC). ADC approaches quickly the good values of HNC and lies just a small percentage beneath its value. Average Hops Rate shows HC outperforms slightly better than HNC. ADC is even able to outperform HC by a small percentage and probably much more in longer test runs. Average Load Variance shows that their adaptive approach very quickly approached an almost optimal value and competes with the good load balance of HC. HNC performed poorly in comparison to HC and their algorithm.

In the second simulation, they ran all three algorithms against a uniform hot spot for a set of 1000 objects. Average Hit Rate showed that load balancing of hot spots was the major strength of the proposed algorithm. ADC was able to maximize the total cache usage through minimization of redundant objects allowing a hit rate of over 90.
One limitation of the algorithm lies in the fact that it keeps a mapping table record for each requested object, which can lead to a large amount of system data.

[6] introduced in their latest work an algorithm for adaptive distributed caching (ADC) based on a set of autonomous proxy agents. There algorithm was well able to find a balance between optimal cache usage and distribution of hot documents and reached good values for the object allocation results that are comparable with the near-ideal hashing algorithm. The core of the ADC algorithm can be divided into two local techniques that allow global stabilization: Request Forwarding and Selective Caching with its components Mapping Table and Self-Organization by Multi-Casting. The same version of the algorithm was installed on every running proxy. In their previous work[17] they allowed the table to grow infinitely, keeping track of all previously experienced objects, which usually leads to out of memory problems and performance drawbacks. The single table is used to simply keep track of the current flow of requests. Each unknown object will receive a new entry on the top of the table and displacing the oldest entry at the bottom of the table the well-known LRU algorithm. When an existing entry in the mapping table experiences another hit, the time difference between the two requests can be used as a first approximation of the average object request frequency and the object can move from the single table to the multiple table. The LAST column represents the local time value, for the last time when the specific object was requested while the AVG value represents the average time between two requests of the same type.

OBJ-ID	PROXY	LAST	AVG	HITS
www.xy634	Proxy[5]	9952	0	1
www.xy34	Proxy[4]	9953	0	1
www.xy123	Proxy[1]	9954	0	1
www.xy64	Proxy[2]	9955	0	1
www.xy53	Proxy[1]	9956	123	432
www.xy343	Proxy[7]	9957	0	1
www.xy29	Proxy[4]	9961	0	1

Figure 8: A Sample Single Table

OBJ-ID	PROXY	LAST	AVG	HITS
www.xy64	Proxy[8]	2252	70	2
www.xy55	This	4253	75	2
www.xy13	Proxy[1]	4154	83	34
www.xy644	This	6555	90	2
www.xy52	Proxy[4]	3356	123	42
www.xy433	Proxy[8]	7557	313	4
www.xy299	Proxy[4]	3261	874	54

Figure 9: A sample Multiple Table

The multiple table is also restricted in its size and contains only objects that were requested more than once ordered by their average request time. Removed objects from the multiple table will be placed into the single table as a regular entry, giving it the chance to be hit again later. The cached table in a proxy within the ADC architecture keeps track of all currently cached objects there. This table is very similar to the previously described multiple table, with the exception that the table entries represent actually stored objects.

For their own ZIPF distribution, the ADC algorithm was well able to compete with both hashing approaches. For the second session of experimentations, they tried to create artificial data that represents closer the request patterns

experienced on a real system and used the widely used Polygraph tool to build a file with 2.3 million requests. It was clear that the updated version of the ADC algorithm (with single, multiple and caching tables) outperformed both hashing algorithms over the set of 2.3 million requests. In the second part of this section they take a look at the average hops rate, which represents the effort to search for a specific cached object. One hop is defined by the request being forwarded from one proxy to another one towards the target server and on its way back. Their results showed that both hashing approaches need on average around 5.5 hops to resolve a requested object.

1) Performance and Parameter Sensitivity of ADC: ADC allows the distributed proxies to agree on the specific location of one object without the need for a central coordinator or a broadcasting protocol. In hierarchical and hashing systems, every proxy stores all passing objects regardless of its future significance and usually uses the LRU algorithm as the cache replacement strategy. This approach has the drawback that it creates a high cache fluctuation rate with minimal reliability in regard to the cached content. Proxy agents based on ADC keep track of the average request frequency of all requested objects based on the last two requests experienced. The learned data, in the form of time gap between two requests, will be used to decide whether the new data should be cached or not. A newly arrived object will only be cached if its average request time is smaller than the worst case currently residing in the cache. Thus they, introduced selective caching as a mean to focus on the more important often requested objects, and preliminary work has shown that ADC algorithm works better with selective caching and an ordered table than a table based on a typical LRU algorithm. An object will only be cached if it is able to move from the multiple-table into the caching table by having an average request time shorter than the worst case. To make sure that old objects will expire, they introduced a simple object aging strategy by computing the object average time with a focus on the passed time since the last request.

$T_{age} = (T_{average} + (T_{now} - T_{last}))/ 2$
The advantage of this equation is that it is simple and incurs minimal computational cost.
a) Experiments: To validate and verify the described algorithm for adaptive distributed caching, the author compared the results of the ADC algorithm to the widely used distributed caching based on a hashing algorithm and evaluated the performance in regard to hits rate, hop count and execution time.

ADC versus Hashing
They compare the performance of ADC algorithm to the performance of a common hashing algorithm using following parameters:
It is clear that the ADC algorithm drags after the hashing algorithm to reach its high values in fill phase of the polygraph dataset, but is then able to outperform the hashing algorithm by a small margin.
A hop is regarded as the message transfer between client-proxy, proxy-proxy and proxy-server. They can safely claim that on average, the ADC algorithm needs two more hops than the hashing algorithm to resolve an incoming request.
The results show that this version of ADC that employs multiple mapping tables requires a learning period before it can match the performance of the common hashing algorithm. However, the performance of ADC is fairly stable with respect to the size of the mapping tables.

2) Adaptive Caching Techniques in peer-peer Network: On the Internet, peer-to-peer is a type of transient Internet network that allows a group of computer users with the same networking program to connect with each other and directly access files from one another's hard drives.
There are two types of caching used in the peer-to-peer systems: adaptive and dynamic. An adaptive caching consists of multiple distributed caches which dynamically join and leave cache groups based on content demand. Dynamic Caching is a caching technique that caches objects that change often and very infrequently.

a) Squirrel
Squirrel caching technique evaluates decentralized web caching algorithm. The key idea in Squirrel is that each client browser exports its cache to other nodes in the corporate network, thus synthesizing a large shared virtual web cache. Squirrel pools resources from many desktop machines which are called clients.
A typical Home-node approach in the Squirrel caching technique: When the requested object is not found in the local cache, the request is passed to the home node. If the object is there it is dispatched to the requesting client (A). Otherwise, the request is passed to the origin server (B1), and the origin server replies with a not modified message, or the object to the home-node (B2). Then the home-node sends the object to the requesting client.

b) Tuxedo
It includes an adaptive neighborhood set algorithm for different Web servers, and a hierarchical cache digest for sharing of transcoded versions and value-added services. Tuxedo is more scalable and completely decentralized than other peer to peer caching techniques. Tuxedo is an overlay network which uses efficient neighborhood propagation and adaptive neighborhood mechanism for different web servers, and by using a hierarchical cache digest to store information related to transcoded content.

Two tables maintained at each Tuxedo cache node: server table and neighbor table. Using the information at the server table the peer S1 points to three peers P1, P2 and P3 whose latency, bandwidth, cache digest and reputation information is given in the neighbor table. So, when a request comes to S1, it points to these three peers from where the contents can be fetched.

c) PeerOLAP

PeerOLAP is developed for distributed caching of supporting On-Line Analytical Processing (OLAP) queries. If a query cannot be answered locally, it is propagated through the network until a peer that has cached. The PeerOLAP network is a set of peers that have local caches and implements a mechanism for publishing their local caches contents and their conceptual capabilities. The peers access data warehouses and pose OLAP queries. Other peers can connect to the peer Pi and request a result. The goal of PeerOLAP is to act as a combined virtual cache, where all the components offer resources aiming at achieving lower query cost.

A typical PeerOLAP network. Peer P2 issues query of referring to chunks C1, C2 and C3. C1 resides in P2; so it sends message requesting for C2 and C3 to neighbors P1, P3 and at the same time to another node P6. The peers who have any of the chunks, sends cost estimation of retrieving to P2.

From above, it is found that there are some efficient ways to improve the caching technique: -The cache will be shared from the memory of the peers. One found it inefficient to share the cache of the browser. -The cached object will be stored in chunk; it ensures the availability of the peer for a longer time. It also takes less space in a peer's memory. -The peer or delegate choosing criteria will be based on bandwidth or latency.Thus it can be used to find out the fastest content residing peer in this way, although it has the drawback that it doesnt support multiple servers and the cost of fetching the object may be high

D. Application of Distributed Cache

Adaptive Caching Mechanism for Web Services: Web services, with an emphasis on open standards and flexibility, can provide benefits over existing capital markets integration practices. [5] proposed a novel adaptive cache mechanism, which can choose an optimized cache implementation dynamically in the runtime. They presented a fine-grained cache approach to obtain the further performance gain and design a cache key associated method to decrease the usage of memory. Through experiments it was observed that their approach obtained a huge performance gain by incorporating the adaptive cache mechanism into the SOAP engine. SOAP is based on XML technology. XML parsing and deserialization is exactly the performance bottleneck of Web services. Based on the analysis, they proposed an adaptive cache mechanism, which can choose an optimized cache implementation dynamically in the runtime for request. Moreover, they used many novel mechanisms to optimize cache, such as fine-grained cache approach and cache key associated method. They implement these strategies in SOAP engine and the experiment results showed that the approach was efficient and improved the performance of Web services remarkably. The client-side cache has some inherent drawbacks, e.g., low hit ratio and difficult to manage consistency, which make it can't be used in some situations. Compare to client-side cache, server-side cache has the higher hit ratio and is easier to manage consistency. These are reverse proxy cache, which save the response message for the follow requests. However, the cache key is associated with parameter value, so the hit ratio is very low for some Web services. Thereby this cache implementation cannot be widely used. In this paper, they saved every object include complex type and its fields, for application. They called this method as fine-grained cache mechanism, which are most useful when the reappearing probability of the fields of complex type have a striking dissimilarity to each other. In application object cache implementation, the key can be generated pre-parsing and post-parsing. SOAP Express uses XML Pull Parsing technology to parse XML element. The application pulls the XML event from XML Pull Parser, and gets the event of XML elements sequentially. It only goes through XML data once. The fastest way is to use the text as the key directly, but a major problem is that a semantically identical XML request message can be represented in several ways.

The mediator component can map the semantically identical XML element to the same cache key. Using the mediator component, they can generate cache key efficiently without XML parsing and avoid saving the same cache content. If the mediator return the null object, which means the cache missing. If the cache key is generated pre-parsing, the whole time needed to process the SOAP message is shown as follows: $T_{pre}=t_m+t_s+(1-\alpha)(t_p+t_r)+t_{other}$. If the cache key is generated post-parsing, the whole time is: $T_{post}=t_p+t_s+t_{other}$ Where α is the hit ratio, tm is the time that mediator used to get the cache key, t_s is the cache searching time, t_r is the cache registering time, t_p is the time used to parse the XML element and t_{other} is the time used to do other processing. $\theta=(t_m+t_r)(t_r+t_p)$. SOAP express implement three cache methods: response cache, objects cache1 (generates cache key before XML parsing), object cache2 (generates cache key after XML parsing). Using adaptive cache mechanism, they combined the three cache implementations together and choose a better one for each request. This approach can utilize the predominance and avoid the drawback of each cache methods.

In their experiment, they gathered two groups data: average response time and transactions per second. The parameters of the services are as follows: null, simple type, array type and complex type. They found that average response time of Delta SOAP Express is much lower than that of SOAP Express and the transactions per second is higher too. The average response time of Delta SOAP Express is much lower than that of SOAP Express and the transactions per second is higher too. They saw that when the hit ratio is greater than 75% (approximately), the response cache is the best while when hit ratio is between 31% (approximately) and 75%, the object cache 1 is the best, and in other situation, object cache 2 is the best. Using adaptive cache mechanism can choose the best cache implementation for a special situation in the runtime, therefore this mechanism can help server to get the most performance improvement. Experiments show that the adaptive cache mechanism is efficient and feasible.

2) Adaptive Generation of Caching in Cellular Mobile Networks: In such a mobile computing environment, mobile hosts move from one place to another and need to access the information without interruption. That is why location management is one of the most important issues in mobile computing environment. [13] studied how to generate location caching for systems using tree-structured databases. An adaptive location caching generation method was proposed to calculate the optimal caching nodes to minimize the total location management cost.

In a cellular mobile network, the whole service area is divided into a collection of inter-connected cells. Mobile clients may move within their current cells or move into other cells. The mobile computing system has to maintain the real-time locations of moving clients while they are moving. In the existing cellular mobile networks, a two-tier location database architecture is adopted to manage the locations of mobile clients. This two-tier architecture is simple and easy to implement but has some drawbacks. The number of mobile clients in a mobile computing system can be very large, thus the two-tier architecture is obviously not scalable. Secondly, because a mobile client is permanently associated with a home database termed Home Location Register (HLR), the overheads for maintaining the locations of mobile clients can be very heavy if the mobility of clients is high. The procedure of their [13] proposed adaptive caching strategy can be divided into two steps: Firstly, those cells which have more communication with each other are divided into one group A. Then, an effective split algorithm is used to calculate the caching nodes under group A. To the best of their best knowledge, it is lack of any detailed study on the adaptive caching method based on the hierarchical database structure. [13] proposed an adaptive caching strategy with the objective to reduce the total locating cost based on the call patterns of clients. The main idea is to maintain a call matrix, C, to count the number of calls between two cells to track the changes in the call patterns of clients. Whenever a mobile client from cell i calls for the other from cell j, C in the matrix is increased by one.

Database Suppose the location databases are organized as shown in Fig. 21 and Fig. 22 table shows the call matrix for the location databases. From the call matrix, we can see that mobile clients receive more calls among Cells 4, 5 and 7, than other cells. However, the two location databases responsible for Cells 4 and 5 form a cluster with the location database responsible for Cell 6. Thus the total location management cost is not optimal. The call matrix of a location database system, as shown in Table, consists of nine cells. Each entry in the matrix indicates the number of calls between the corresponding two cells. For example, the value 110 of the entry C1, 2 indicates that there are 110 calls between Cells 1 and 2. Based on the number of calls between cells, they cluster the location databases in the location database tree to minimize the total locating cost. The cells that have more calls are clustered in the same unit. In order to find the best way to group cells, they calculated the call affinity of cells. They exchange the rows and columns in the matrix to calculate CAC for different arrangements of cells. They choose the matrix with the greatest value of CAC for clustering. And further- more the matrix is divided into several parts from up to down with each part's size smaller than the predefined size threshold. After the cells are clustered, they have to decide how to set the caching nodes based on the clustered unit. Assume V is a set of cells after clustering, and Vi which contained in V represents a cell in the service area, corresponding to a leaf node in the hierarchical database structure. The question is set the caching nodes based on clustered unit V.

They proposed algorithm Split(Unit V) to separate a certain unit V. The input of their algorithm is a set of cells V which its size no more than a predefined threshold u.
The output of their algorithm is two sets that satisfy the conditions given above. They plotted the locating cost when two different update generation policies are adopted: the distance-based method and the cost-based method. It was observed that the locating cost under adaptive caching method was much smaller than easy caching, especially when the RCMR was large. The overall locating cost under cost-based update method was better than distance-based method due to a smaller number of location updates.

3) Adaptive Caching Scheme for Heterogeneous Mobile Devices and Wireless Networks: a Service Oriented Perspective: The 3G environment surely of heterogeneous mobile devices and heterogeneous wireless networks. Users can carry heterogeneous mobile devices with different cache sizes, transmission range, and even transmission latency. These devices not only can get services from different Mobile Support Stations (MSS), but also can form a mobile peer-to-peer (P2P) network to provide services to each other. For such a complex mobile network environment, an effective

cache mechanism that can handle the heterogeneous properties is required. [14] proposed a flexible cache scheme which is adaptive to the actual device condition and that of its surrounding environment.

Lots of cooperative caching algorithms have been designed for distributed network caching. In the CUFSR (Cooperative Uniform Frequency, Size, and Recentness) caching algorithm, it considers Frequency, Size, and Recentness as the main factors in designing the cache updating scheme, in which a node can get data from anothers cache; in this scheme, nodes have no difference with respect to the caching scheme adopted. In DLRU (Distributed Least Recently Used), the caching updating scheme uses LRU (Least Recent Used) as its updating algorithm, and nodes can interact in a cooperative way for data sharing. [14] proposed an adaptive caching scheme catering for the different properties of mobile devices. They suggested an AFSR (Adaptive Frequency, Size, Recentness) scheme, and conduct analytical studies to compare it with the existing schemes of CUFSR and the popular DLRU. Simulation and mathematic analysis results showed that their scheme is more effective.

To address these challenges, they advocated an adaptive caching framework[14] to cope with the heterogeneous devices with different transmission ranges, latency, and even cache size, coexisting in mobile P2P network. For each data request, a mobile host mi first tries to find the required data item from its local cache. If it encounters a local cache miss, it uses the Cache Path to find if the other peers have the required files. If a neighbour peer has the required data item cached, it sends the required message to mi via P2P communication. If mi cannot find the required data from the Cache Path, it broadcasts a request message to its neighbor peers via P2P broadcast communication. However, if mi still does not get any reply from the P2P network after the time out period, it will send the request to a particular MSS instead which has the required data and is of the least latency to provide the required data. As all mobile devices have two network interfaces, they can use one of them to communicate with their neighbors. If a peer in the P2P network can get files from other peers directly or through some relaying peers, then all of these peers will be regarded as neighbors, with the data transmission latency defined below:

$\sum_{i=1}^{hop}$ (size of request$_i$+ size of reply$_i$)/BW $_{i,i+1}$

Their service based architecture incorporated a number of facilities and techniques are shown in figure 4.6. In fact, techniques such as Cache Data, Cache Path and Data Replication are some of the most popular ones used in caching. From an individual peers viewpoint, it stores different properties about their neighbour nodes, and can get services from the other mobile devices as well as from the MSS.
In the first part, the most suitable data representation method is selected according to the specific application demand, which also facilitates the usage of the caching scheme. In the second part, different caching techniques can be combined according to the application need; and in the third part, they use machine learning techniques to choose the appropriate caching algorithm and determine the appropriate caching parameters. With such a layer, data usage of the upper level(s) of this overlay network can be greatly facilitated. In their scheme, an important attribute for each data file is its popularity: the larger the value is, the more possible that mobile clients may access the file. Formula (2) defines the relationship between the popularity of a file and its rank, which can be denoted as Zipf-low, where is assumed to be around 0.5. popular= $1/_{rank}$(1<rank<TotalNum)

They employ the FSR scheme as the basic cache updating algorithm for each mobile host, in which the weight of a cached file is defined as: $W=F^fS^s R^r$
F (Frequency) is the number of times the cached file is accessed, S (Size) is the size of the file and R (Recency) is the counter of the required times since the last access to C_i . Each new request will cause all the cached files recency to be incremented by one; if a cached file has been actually accessed, its recency is reset to zero. Also they should choose cached files with small F and large S values as the candidates for replacement, so that it may allow us to accommodate more hot yet small-sized files in the available cache space.
Consequently, mobile devices are divided into: 1. Strong nodes (with Low Latency, Big Cache Size and Big Transmission Range), 2. Weak nodes (with High Latency, Small Cache Size and Small Transmission Range) and the rest into the third group as Normal nodes. When a data request is received by a particular node, an efficient routing algorithm is invoked to generate the nearest path to the file source peer. In fact, every node keeps a bitmap as shown in Table 1 with the following policy: if there is a cached file existing in a particular node, then the number in its bitmap is set as 1; else 0. Consequently, the overhead incurred is rather small.

Cache/Node	Node1	Node2	Node3	Node4
File1	1	0	0	0
File2	0	1	1	0
File3	0	1	0	0

In their approach, they choose the special path with the minimum total latency to get the required data. So, if more than one source node exist, they should compare their weights to make the choice.

Adaptive Data Caching Scheme

Upon receiving a data request, the caching algorithm checks if the required file can be satisfied locally, from its neighbours, or through a particular MSS. More specifically, if the required file is not in the local cache, the peer (mobile device) checks if the required file is available from any of the other neighbours peers or those recorded by the peers cached path. The overall cache updating scheme is given in Algo1, with the Greedy updating algorithm (line 2), Conservative updating algorithm (line 5) and the Selfish updating algorithm (line 4) given in Algo2, Algo3, Algo4 respectively, the parameter f means the file to be used for updating the cache(Cache List).

In the Greedy cache update approach (for a Strong node), the algorithm attempts to keep most of the hot files in the local cache of the mobile device. Furthermore, the data replication method can be used to keep the most popular file in the Strong peers. In contrast, the updating algorithm shown in Algo3 is for a Normal node, in which the local cache is used to keep those files which may not be the hottest ones but are still likely to be used in future. As a matter of fact, the scheme is Conservative because the nodes do their work by considering their stronger neighbours condition and then decide their cache updating accordingly. In Conservative scheme, a Normal (non-Strong, nor-Weak) node is able to keep the reasonably popular files which do not appear in the Strong neighbours. The reason for this caching strategy is due to the fact that even the Strong nodes are still of limited caching space. As the third possibility (when the node is a weak one), the local cache is updated according to the Selfish scheme. They do not need a Weak node to provide services to any other nodes given their extremely limited ability, so Weak nodes only need to update the cache for their own use. In their caching mechanism, they cluster the Strong nodes and replicate popular data within the cluster. A threshold on the popularity is adopted, so that if a files popularity is larger than the threshold, it will be replicated among its cluster of Strong nodes. By this method, they can distribute the most popular data over the Strong nodes to serve the entire network. In contrast, as the transmission latency of the Weak nodes is large, they use prefetching to forecast the data requests upon them. To implement the prefetching, association rule mining technology is used to predict the (near-)future data requests upon the Weak nodes.

In Algo5, a detailed prefetching algorithm was shown, in which a subsequent file will be prefetched if: (i) the file is immediately following the currently required file, and (ii) the access times of the currently required file exceeds a predefined threshold.

They compared their scheme with such existing traditional algorithms as CUFSR (Cooperative Uniform Frequency, Size, and Recentness based algorithm) and DLRU (Distributed Least Recently Used), the later is especially popular in cooperative web caching.

Notably, AFSR algorithm performs better than CUFSR in all cases, while the latter is already better than DLRU. As node number is small, the predominance of AFSR is much obvious. In fact, when simulation node number is big, then the cache is big enough to contain almost all the files in the cache, so the three algorithms performance is much the same.

ARD gets larger as the file number increases. The author next compared the inner request disposal mechanisms of the three different cache schemes, by recording the total requests disposed with the different kinds of the mobile peers.

Most cache requests in AFSR are accommodated by the Strong nodes, while the least number of cache requests are disposed by the Weak nodes. As for the other two schemes, no consideration has been given on the heterogeneous property of the mobile peers, so the cache requests satisfied by the three kinds of peers are almost the same.

4) A Distributed Cache for Hadoop Distributed File System: In HDFS file reading may contain several interactions of connecting NameNode and DataNodes, which dramatically decreases the access performance when the system is under a heavy burden of data and a concurrent workload. Thus, how to improve HDFS file access performance (especially file reading performance) is a key issue in real-time cloud services. [4] presented a novel distributed cache system named HDCache built on the top of HDFS, which has rich features and high access performance. The cache system provides general purpose storage for variable workloads including both huge and small files. The most mentioned shortage of HDFS is the weak performance when dealing with small files. Combining small files into large files or modifying HDFS I/O can't improve performance. The goal of their work is to design and implement a distributed cache system on the top of HDFS that can accelerate person-specific data access in large-scale real-time cloud services. Their HDCache system is based on the following factors, prerequisites and design considerations. On-the-top Method rather than Built-in Method, Network I/O rather than Disk I/O, Layered Data Accessing Model, Motivations of Designing a New Cache System. The Memcached system is not suitable for Hadoop because of the following: (1) The Memcached system is designed for caching data that are stored in a database. It is not a typical cloud storage system. Memcached uses a pre-allocated memory pool called slab to manage memory. This memory management strategy fits the DB datawell (SQL query results are usually small) but does not fit online personal data whose size is in the range from several KB to dozens of MB. (2) The Memcached has no local serialization or snapshot mechanism. (3) The servers of the Memcached system are independent one another. (4) The Memcached has a simple consistency checking process by setting an expired time. This may cause a burst of network traffic when a large number of cache data are expired simultaneously. Their novel HDCache system is built on the top of the Hadoop Distributed File System. The cache

system and the HDFS are loosely coupled. The system can be viewed as a C/S architecture. The only thing the third-party applications need to do is to integrate with a client side library.

The HDCache system can be deployed on HDFS NameNode, DataNode and any other application systems (such as web servers) that can access HDFS through networks and need cache functions no matter whatever the OS of these systems are Windows or Linux. The cache system contains two parts: a client dynamic library (libcache) and a service daemon. Users only need to integrate with the libcache in their applications, and they can access the cache services that are on the same machine or connected through networks. One cache service can serve multiple applications simultaneously.

The HDFS access module uses HDFS API to load file contents into pre- allocated shared memory. Shared Memory Manager (SMM) is the core module that bridges the service and client library. The serialization module periodically writes the meta-data and all cached files into a local OS file system, forming a series of snapshots which can be used for reestablishing a cache after a system crashes down. Another function of the serialization module is to provide swap when the cache is deployed on the small RAM hosts. Compared with highly efficient shared memory, they choose sockets as an inter-process communication facility for the reason that libcache can contact with both local and remote cache services in the same way on heterogeneous systems. The consistency of cache is guaranteed by a validator which has comprehensive validation rules in order to achieve the balance of resources consumption and timeliness. The HDCache system provides a client library called libcache integrated by upper layer applications. The libcache consists of two major components:

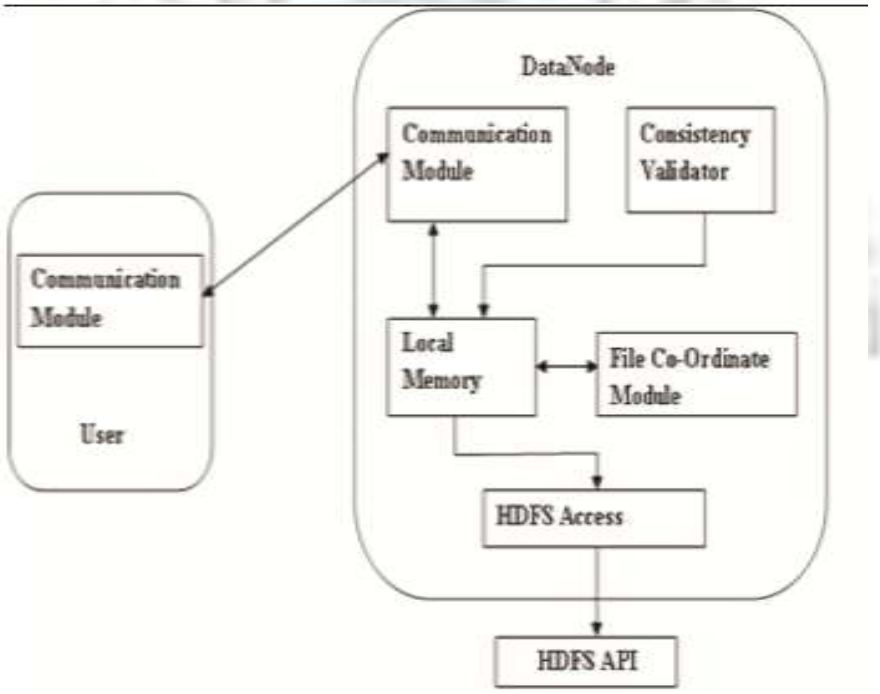

Figure 10: Hadoop

Communication and Control and Shared Memory Access. The Communication and Control module undertakes the tasks of (1) interoperating with the HDCache service on the same host, (2) communicating with ZooKeeper servers remotely, and (3) calculating hash values of desired files and locating a specific cached file. Compared with the Memcached system in which client applications copy the cached content from a service process to their own process memory spaces, their cache system chooses shared memory which theoretically has higher performance. When a client opens a cached file, the libcache records the first page address of this file and returns a file descriptor to upper layer applications. The client uses this file descriptor to read/write the content of the file.

When the client acquires a file and the cache misses, the cache service will fetch the file from other cache services, local snapshots or HDFS. When the cache service loads the fetched file to its shared memory, if the memory has no enough pages to be allocated, the service will use LRU (Least Recently Used) algorithm to eliminate some files unused for a long time. The statistic information for the LRU algorithm is also stored in the Meta Info Map. When processing LRU in memory, the eliminated file can be swapped into a local disk. If the local disk has not enough space, LRU also can be applied to a local disk space to delete some unused files. If the expire time fires, the consistency validator decides whether to process consistency check according to network traffic, system workload, and the file access frequency.

When a large number of files expire at the same time, a random delay is added to the consistency checkpoint in order to reduce an unexpected burst of network traffic. HDFS introduces replica to enhance system robustness. By default, each file stored in HDFS has three replicas that are stored in different machines, which minimizes the impact of a machine crash. Their cache system also adopts this design that every file cached has three replicas. Compared with the HDFS single NameNode design scheme, their cache chooses the DHT (Distributed Hash Table) instead.

System Operation	Operation Time(milisecond)
REDIS GET(10 MB)	12.61
REDIS GET(10 MB)	17.13
REDIS GET(4KB)	0.039
REDIS GET(4KB)	0.051
Memcached GET(4KB)	0.034
Memcache SET(4KB)	0.038

Table shows experimental results of Redis and Memcached. The results indicate that when processing small files the three systems almost equally performed, and when dealing with big files, their cache gains the better performance.
The read efficiency declines with the increase of client threads. However, when the number of threads exceeds 90, the efficiency fluctuates in a constant range. The read efficiency can keep greater than 1GB/s when the number ofclient threads is up to 200.
The response time of the cache fopen operation is prolonged with the number increase of client threads and fluctuates in a constant range when threads exceed 30. The cache fopen operation can keep within 10ms when the number of client threads is up to 200. Concurrent tests show that their cache system can maintain very high performance in both services and client library in multithread environments.

5) Memory Management Architecture for Mobile Computing Environment: [3] proposes a common memory management mechanism for mobile terminals and servers called Memory Management Architecture for Mobile Computing Environment (MMM). MMM allocates a part of the memory of a mobile terminal and a part of the memory of a server as common memory and maintains the consistency of the common memory areas. MMM can reduce wireless traffic to maintain consistency, and that depending on the charge policy, a different pre-fetching scheme is efficient in minimizing communication cost. It is hard to develop application programs that co-operate in mobile computing environments because of the complicated nature of wireless communications. MMM controls a part of the memory of a mobile terminal and a part of the memory of a server which are common to each other.

Purpose of MMM
a) Synchronization of common data between a mobile terminal and a server.
b) Easy programming of applications with the cooperation of a mobile terminal and a server.
c) The increase of processing speed, the reduction of communication cost and power reduction.

MMM is similar to cache memory as
a) Current data used are fetched from the memory at a remote location and stored in local memory.
b) MMM is a kind of distributed shared memory in which some common address spaces at physically different locations are controlled to have a shared address.
c) It provides virtual memory space to mobile terminals.

The external memory for virtual memory corresponds to a common memory space on the server.

Memory Management Systems
1) Memory lines and memory control status field
The common memory area of a mobile terminal and a server is divided into fixed areas called lines. The memory mapping is handled by line units. A line is divided into several blocks, and memory write back for memory consistency is handled by the blocks. The memory control status memory is introduced to control the memory line status.

2) Memory control status field and memory line status
Memory line status is determined by the memory control status field. Each memory control status field is composed of V bit, C bit and plural D bits.
Vm, bit in a memory control status field on a mobile terminal is a valid bit which indicates whether the corresponding line is valid or invalid. Cs, bit is a copy bit which indicates whether the line has been copied to a server or not. Dm, and Ds, bits on a mobile terminal and a server are dirty bits that show whether the write operation has been performed in those blocks.

3) Memory access and line transfer control
When a memory access occurs, the kind of memory operation and the succeeding memory line status are determined by the kind of memory access and the current memory line status. The line transfer control and related memory line status control are performed in a memory exception interrupt which is invoked by a memory access.

4) Resolution of line access conflict
When a mobile terminal and a server request the same line at the same time, there is a possibility of memory deadlock. Preventing the deadlock, to accept the remote interrupt is inhibited between the acceptance of memory exception interrupt and the completion of the first instruction execution after the end of the interruption. By this method when the same line is used, the memory line is accessed alternately between the mobile terminal and the server.

5) Control of communication failure

If a memory exception interrupt occurs, the connection of wireless communication should be established. And if the connection fails, the control program reports the status of memory to the application. The application can use the data conditionally. By means that a common memory area on a mobile terminal is controlled by virtual memory, memory on a mobile terminal can be expanded. Virtual memory of the common memory area on a mobile terminal is mapped to real memory, and the necessary memory for the time being is located in real memory. Virtual memory and real memory are divided into pages. Virtual memory uses pages as a base of control. If the line size is different from the page size and is smaller than the page size, the page size should be equal to the integer times of the line size. Similarly, if the line size is larger than the page size, the line size should be equal to the integer times of the page size.

Pre-fetching of line: Increasing the line size increases wait time and reduces usability, but if the program execution and the line transfer are performed in parallel, it is possible to enhance the execution speed. This can be realized to pre-fetch the lines that are probably used next after the requested line is transferred. It is possible to pre-fetch multiple lines. But if too many lines are pre-fetched, unused data increase and the overhead increases. The effect of pre-fetching is proportional to concurrency of program execution and line transfer.

MMM (Memory Management architecture for Mobile computing environment) is the most preferable memory architecture for mobile terminals and for servers. It incorporates cache, virtual memory and shared memory architecture. The efficiency of the communications could increase by pre-fetching multiple lines.

6) Memcached with bluetooth automated distributed caching: In [12], the caching has been used to finally improve the data accessibility of mobile devices.

Autonomic system
The autonomic computing community enabling autonomic behaviour for systems. Free up system: Administrators from the tasks of low level system administration optimization. An autonomic manager can be functionally decomposed into multiple stages of a control loop called the MAPE model. The Managed Resource is simply the system that is to be adapted, the assumption being that it is not initially adaptive and that it can be modified to support run-time adaptation. The MAPE model requires two types of touch-points into and out of the Managed Resource Sensors and Effectors. Sensors to collect data from the Managed Resource, and Effectors to perform adaptations to the Managed Resource.
These are the functions that make up the decision process needed to adapt a system. Monitoring uses the Sensors to produce monitoring events for consumption by Analysis. Analysis processes monitoring events to look for potential problems and opportunities. Planning decides, based on the system state and analysis messages, if it is appropriate to perform an adaptation. Execution utilizes the Effectors to perform the adaptations requested by Planning.

Characteristics of AC

Self-configuring: Autonomic systems will configure themselves automatically in accordance with high-level policies representing business-level objectives. Self-optimisation: Autonomic systems will continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost.

Self-protecting: Autonomic systems will be self-protecting in two senses. They will defend the system as a whole against large- scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures. They also will anticipate problems based on early reports from sensors and take steps to avoid or mitigate them.

Self-healing: Autonomic computing systems will detect, diagnose, and repair localized problems resulting from bugs or failures in software and hardware, perhaps through a regression tester.

Mobile computing environment Augmenting Autonomic Manager features into mobile computing environment:- Consider a mobile environment with n cells C1, C2,.... For each cell Ci, DSi is the database server that can keep pieces of information that may be accessed by other systems. A client is a system, which invokes queries for data. Each cell Ci contains a set of clients S1, S2.... Each client S1 of the cell C1 can issue the query through the base station BSi which is directly connected to the database server DSi properties. Systems with self-healing capabilities are able to remain functional by reconfiguring or repairing themselves in response to system faults, potentially increasing their fault-tolerance and reliability. Memcached is a promising mechanism that could be applied to create an autonomic mobile computing environment through which devices can self-heal in response to limited cellular connectivity. Memcached is a distributed data caching mechanism that uses hashing to map data A database server can contain more than one database and can indirectly communicate with all mobile clients in the same cell through the Base station BSi. Autonomic computing is computing paradigm in which systems have the ability to self-manage without intervention by an outside entity. When data stored with Memcached is needed, a requesting node can use the hash function to retrieve data either locally, if a local copy exists, or from another node that it has been stored on.

Mobile Distributed Caching with M2Blue

Mobile Memcached with Bluetooth (M2Blue) is an autonomic distributed data caching mechanism created to allow mobile devices continued access to data by using autonomic distributed caching to self-heal in the event of limited cellular connectivity and provide a mechanism for caching data that can self-heal to function despite the entry/ exit of devices from the network.

Autonomic Device Modes: Read/Write/Store
Store mode: While all devices should be able to read from and write to the cache, all devices are not always used to store cached data. This state is represented as Boolean variable named store mode.

Write mode: The write operation defines how data is written into the autonomic distributed cache when the key, data, and data tag are received by a device from the server. Once a predefined threshold of reduced connectivity is reached, a device, referred to as the initiator, will begin the self- healing process by autonomically invoking the begin command. The initiator uses Bluetooth to detect the presence of nearby devices and then transmit a packet containing its device ID and a list of detected device IDs to the server over the cellular connection. Upon receiving this information, the server will reply to the initiator with a packet containing the public key and a hash function. The initiator will then use Bluetooth to broadcast the public key and hash function to all discovered devices, creating a mobile network profile that can be used in conjunction with autonomic distributed caching to allow self-healing in the face of limited connectivity.

7) Resource Efficient Adaptive Caching Scheme for ad-hoc mobile networks: In a Mobile Ad-Hoc Network (MANET), wireless devices have limited resources. Storage space and battery life are two of these resources that must be managed efficiently in order to extend the usefulness and lifespan of the wireless device. Cooperative caching in ad-hoc networks was developed as a method where the storage burden is more evenly distributed, or shared, across the entire network. This scheme helps to alleviate some of the negative impacts of basic caching.[10] proposed a novel scheme that seeks to distribute the storage, bandwidth and energy burden through a resource-efficient adaptive caching scheme for MANETs. Their approach to this problem introduces significant resource efficiencies by implementing an adaptive cache distribution and data replication scheme(Tidal replication).This is accomplished by provisionally replicating specific data items towards the branches of the network that have a sudden and disproportionate demand. When a data item is discovered with a statistically significant weighted bandwidth utilization, the node flags that data item for replication. The node then sends a replication request out towards the Dominant Request Path (DRP) for that data item. The DRP is defined as the network path that is responsible for the majority of requests for a particular data item at the given node. In Tidal Replication, a tide of demand will cause the replication of a data item from the control node towards the point of a sudden increase in demand. As that demand wanes, the replicas are then de-allocated and the burden for the data item returns to the control node. Magnetic Distribution, aims to dynamically redistribute cached items to a point of equilibrium within the network the Magnetic Distribution scheme will shift control of a specific data item to another node that has a more centralized demand for the data item. Benefits of the scheme is a reduction in the retransmissions of the high-demand data items because of their network demand centrality, reduced cache misses and increased availability of popular data items due to being cached on nodes that are more highly connected and central to the network.

Differences between other schemes and this one: All the other schemes(Cache data,Group Caching etc.) are proactive in caching of replicas, while the schemes in [10] are reactive in order to balance power consumption, energy utilization, and storage space for the network as a whole. In other words, the schemes reviewed took a more localized view when making caching decisions, while the schemes presented in this paper take a more global view.

Tidal Replication

The replication of data items within Tidal Replication is initiated strictly at the Master Data Item Control Node (DICN). The DICN is defined as the node that holds and controls the authoritative copy of a data item. The primary purpose of Tidal Replication is to relieve the bandwidth burden on the DICN due to a temporary flood of requests for a specific data item. This is accomplished by replicating the data items that consume a significant portion of the total bandwidth utilization, for a given control node, out towards the dominant requesting path. The active request window (ARW) threshold represents a slice of time in order to determine what information on the requests is current, as well as to filter out, or remove, what information is not.

Algorithm Details

The ARW threshold, is used to form a weighted bandwidth utilization matrix where each active data point is evaluated based on the total weighted bandwidth utilization over the windowed, ARW, time. Any data point that has total weighted bandwidth utilization that falls positive of one standard deviation from the norm is then replicated towards the DRP.

The formula for weighted bandwidth utilization

$u_i = r_i l_i s_i$

Where,

r_i - the number of requests for a data item,

l_i- mean request path length,

s_i -size of the data item in kilobytes, for the given data point i.

Mean weighted bandwidth utilization for all data points held at the control node:

$u=\sum_{i=1}^{c} u_i / c$

Standard deviation

$$\sigma = \sqrt{\frac{\sum_{i=1}^{c}(u_i - u)^2}{c}}$$

In order to identify a candidate for replication, subtract (+ u) from the weighted bandwidth utilization for each data item. If the result of this calculation is 0 or greater than 0, indicating a weighted bandwidth utilization one standard deviation above the norm, a Replication Request is then sent out to the root node on the DRP. Once the replication request is received, the node evaluates its replica storage availability ratio (RSA) in order to determine if it is able to cache the request.

$RSA=\frac{s-m-c}{s-m}$

The replica storage availability ratio would represent what portion of the remaining space is still available for caching replicas. If the ratio is greater than a given threshold, the node is considered to be able to cache a replica. If it is less than the given threshold, the node must Make Room for the new replica by deleting the replica closest to the TTL expiration. The request matrix is used for the new node to determine if the data item should be cached or redistributed. If the request matrix is in Equilibrium, meaning all requests paths have a balanced request count, then it continues to Cache the replica. If it is determined, through the use of the request matrix, that there is a dominant requesting path, the replica is redistributed down the DRP until it reaches a node at which it achieves equilibrium with its request matrix.

When the replica has been cached at a given node, any requests coming through that node will not be routed through but rather answered directly with the data item. This, in turn, reduces hops, reduces response time, and saves energy through the reduction of the need to rebroadcast both data item and request over many different nodes.

Replica Deallocation

Deallocation methodology utilizes the (ARW) for the node on which the replica is cached. Current utilization,

$U=\frac{x_i}{z_i}$

A maximum age threshold will need to be determined, or arbitrarily set, in order to measure the prune point of the replica. An age based deallocation threshold $T_{age} = \frac{y_i}{z_i}$

TTL, for the given replica

$TTL=(\frac{x_i}{y_i}y)=...=(\frac{y(xi+zi)}{zi})$

utilization-adapted time threshold must be turned into a ratio by which the utility of the replica over time can be determined, called adaptive age threshold ratio T_{ratio}.

$T_{ratio}=(\frac{y(xi+zi)}{zi})/z_i)=...=(\frac{y(xi+zi)}{zi^2}))$

Upon discovering a replica to be de-allocated, the caching node sends a de-allocation authorization request to the DICN. The DICN then replies with an ok message upon receipt of the prune authorization request. If the DICN does not reply to the node requesting to prune a replica after a given time, the node holding the replica assumes DICN status for the data item. The new DICN then periodically checks for the existence of the original DICN, and when original DICN is discovered, promptly demotes itself and repeats the de-allocation authorization request.

Magnetic Distribution

Magnetic Distribution is a process that is initiated at the DICN. The DICN evaluates, on a periodic basis, the Sustained Weighted Demand for each data item it holds. When the evaluation procedure is initiated on the DICN, the DICN uses the historical request data to calculate sustained weighted demand using a Sustained Request Window (SRW).

The method for determining the sustained request weight ratio Δk, $\Delta k=\frac{\sum_{i=1}^{n} c_i}{c_k}$

II. Conclusion

In this paper, we discussed about adaptive distributed cache, distributed internet cache and its various applications. It is concluded that the demand of internet is growing at an exponential rate but unfortunately, the resources available to service this demand are not expanding at a comparative rate. Thus caching is used to overcome the network traffic and this introduces the concept of distributed cache. It is also concluded that server-side invalidation is highly recommended for small-size networks as it is guaranteed to never return a stale object. We discussed about an efficient scheme for increasing the cache hit-ratio in a loosely-coupled cluster, where proxies share cacheable content. The concept of Last-Copy policy increases the number of web items, available at the cluster, and therefore increases the cache hit ratios.

We analysed an adaptive algorithm for distributed cache that combined the advantages from both ideal mapping in HNC with no multiple copies and load balancing through redundant copies found in ADC. This paper also reviewed an ADC algorithm so that it was capable of performing under resource restrictions like physical RAM and discusses a fine grained cache approach and cache key associated method to obtain a performance gain into the SOAP engine. IT also shows how to improve the system performance over a hierarchical location database structure for mobile P2P network with heterogeneous devices and MSS. A greedy scheme was studied to encourage the strong nodes to provide more caching scheme to other nodes, a selfish scheme for weak nodes and a conservative scheme for the rest of the nodes. The results show that this version of ADC that employs multiple mapping tables requires a learning period before it can match the performance of the common hashing algorithm.

MMM is evaluated and it is the most preferable memory architecture for mobile terminals and for servers. We proposed schemes for both replication and distribution of cached data in order to handle resource constraints in a MANET. We also discussed dynamic caching techniques and tried to found some solutions on caching of objects. By combining the advantages of hierarchical and distributed cache, achieving reduced connection and transmission time. Amongst all the concurrency schemes discussed, single thread model showed worst case performance not utilizing the multi-CPU system. Scheme Write-Behind approach performed the best when the ratio of no of read request to no of write request is high, Locking with write preferred over read performed better when there are large read request and fewer write requests.

Also, the dynamic nature of mobile networks, with devices entering and exiting unpredictably, makes it hard to apply self-healing communication with autonomic distributed caching mechanisms, M2Blue autonomic distributed caching mechanism helps mobile networks overcome these challenges and self-heal despite limited cellular connectivity.

References

[1]. Pooja Kohli and Rada Chirkova, Cache Invalidation and Update Propa- gation in Distributed Caches
[2]. Dean Povey and john harrison, A Distributed Internet Cache
[3]. Shigemori Yokoyama, Takahiro Okuda,Tadanori Mizuno and Takashi Watanabe, A Memory management architecture for mobile computing environment
[4]. Jing Zhang, Gongqing Wu, Xuegang Hu, Xindong Wu, A Distributed Cache for hadoop Distributed file system
[5]. Lei Li , Chunlei Niu , Haoran Zheng , Jun Wei Adaptive Caching mechanisms for web services
[6]. Markus J. Kaiser, Kwok Ching Tsui and Jiming Liu Adaptive distributed caching with minimal memory usage
[7]. Kashinath dev rada ,Y.chirkovai Concurrency control in distributed caching
[8]. Reuven Cohen and Itai Dabrai Distributed cache pruning
[9]. Sanjoy Pauly and Zongming Fei Distributed caching with centralised control
[10]. Dan Hirsch and Sanjay madria Resource efficient Adaptive caching scheme for adhoc mobile networksl

[11]. Markus J. Kaiser, Kwok Ching Tsui and Jiming Liu Study of perfor- mance and parameter sensitivity of ADC
[12]. Chandrashekhar Badgujar,Ganesh Dhanokar Memcached with bluetooth Autonomic distributed cache
[13]. Xiaofeng Ding, Yansheng Lu, Xiaochao Ding, Na Zhao, Liang Hong Adaptive Generation of Caching in Cellular Mobile Networks
[14]. Fan Ye, Qing Li, and EnHong Chen Adaptive Caching Scheme for Het- erogeneous Mobile Devices and Wireless Networks: a Service Oriented Perspective
[15]. Md. Tauhiduzzaman, Md. Renesa Nizamee, Sheikh Md. Rubabuddin Osmani,Md. Mohiuddin Khan, A. S. M. Ashique Mahmood Survey on Adaptive Caching Techniques in Peer-to-Peer network
[16]. Pablo Rodriguez,Christian Spanner and Ernst W. Biersack Web Caching architecture Hierarchical and Distributed Caching
[17]. Markus J. Kaiser, Kwok Ching Tsui and Jiming Liu Adaptive Distributed Caching