Towards Cooperative, Self-Organised Replica Management*

David Hales, Andrea Marcozzi
University of Bologna
Dept. of Computer Science
Bologna, Italy
{hales, marcozzi}@cs.unibo.it

Giovanni Cortese
University of Rome
Laboratori di Radiocoms. (RadioLabs)
Roma, Italy
g.cortese@computer.org

Abstract

Nodes in a network often store and serve content to other nodes. However, each has a finite capacity and if requests for content exceed the capacity then queries fail. It is generally not possible for a priori predictions of load demand because at given times some content may suddenly become popular and at others hardly requested at all. Hence over a given time period a population of nodes has a certain total capacity to serve requests (the sum of all individual node capacities) and some demand load (queries going to the nodes). Assuming nodes can replicate content and redirect queries we present a simple node level protocol that self-organises nodes into cooperative clusters leading to efficient outcomes in some simple scenarios. The approach is scalable, robust and self-organising but there are several open issues. We present this as work-in-progress.

1. Introduction

As demand for information increases, centralized servers become a bottleneck. Content providers, system administrators or end users cope with the problem by distributing replicas of content to machines scattered throughout the network. Replica Management refers to the problem of deciding how many replicas of each file to distribute, and where to place them. Enough replicas should exist to handle the cumulative demand but with too few replicas, servers become overloaded, and clients see reduced performance. Conversely, extra replicas waste bandwidth and storage.

Our approach uses a simple protocol that dynamically self-organises clusters of servers that mutually replicate content and redirect queries in order to share load. We have adapted a protocol from a previous abstract model which tested behavior using a simple interaction game (the Prisoner's Dilemma) to capture the contradiction between individual and social goals [1]. We present initial simulation work and discuss what needs to be done to refine the model towards a plausible implementation.

*This work is partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS).

2 The CacheWorld Scenario

We assume a population of N server nodes which form a peer to peer (P2P) overlay network. In addition to being part of the overlay, each node functions as a server responding to requests (queries) from clients outside of the overlay. An example could be that each node is a web server with the overlay linking the servers and clients being web browsers on user machines. Servers store a copy of their own content item (e.g. a website) and have additional storage for k replicated items from other servers. The overlay links servers bidirectionally if they mutually replicate content. In our website example this would mean two linked servers hold a copy of the others' site. We also assume servers have access to three services: a replication service that copies items between servers; a peer sampling service that supplies a random server from the overlay; a content server that serves or redirects queries as required. Figure 1 shows a schematic diagram.

2.1 Query handling

Over a given time period nodes receive queries (load) from clients to serve their content item. Each node has a capacity, C, specifying the total queries it can serve in the given time period. If the load exceeds capacity then the nodes is said to be "overloaded". Overloaded nodes redirect queries to randomly selected neighbors. If a neighbor is not itself overloaded it will serve the query from its local content replica, otherwise it will ignore the query. The essential idea therefore is that overloaded nodes will have neighbors that are not overloaded and can serve queries.

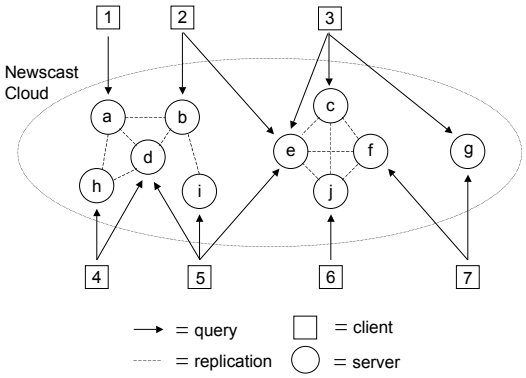

Figure 1. CacheWorld scenario. Server nodes service queries from clients. All server nodes are within a Newscast cloud and may link to other nodes in a P2P overlay. Neighbors mutually replicate content. Overloaded nodes redirect queries neighbors.

```
Passive thread
on receiving a query q, node i:
    if not overloaded, service q directly
    else if neighbors > 0 and q is not
    already a redirected query
        j ← selectRandomNeighbor()
        redirect q to j
    end if

Active thread
periodically each node i:
    if not satisfied
        drop all neighbor links
        j ← selectRandomPeer()
        if j is receptive then link to j
    end if
```

Figure 2. Outline pseudocode for the CacheWorld protocol. The passive thread is activated when a node receives a query, the active thread is activated periodically - once per load cycle.

2.2 Satisfaction and movement

Each node maintains an estimate of the proportion of queries for its own content that are actually served (ps). A node is said to be satisfied when $ps \geq t$, where t is some threshold value. Periodically nodes attempt to change their neighbors (move) in the overlay network if they are not satisfied. When a node i moves, it drops all its current links and selects a random node, j, from the population. i then attempts to link to j. The link is accepted if j is in a *receptive* state. A node is defined as receptive if it is currently not satisfied or if it has spare capacity which is unused. This captures the notion that a node only wants new connections if it is either not satisfied or if it has spare capacity to offer. If a receptive node accepts an incoming link but already has the maximum k links then it drops a randomly selected old link to make space for the new link. Figure 2 gives outline pseudocode for the protocol.

As stated previously, when two nodes link they mutually replicate each others content. Hence movement implies a cost due to the replication process. Currently we do not model this cost in terms of reduction of node capacities. However we do measure the amount of movement that occurs (see later).

3 Simulation Specifics

A simulation model was implemented within the *Peersim*[1] system, an open source P2P systems simulator platform. Our initial experiments have been performed with a small number of nodes $N = 50$. However, we found our results to be broadly scalable. We ran tests up to $N = 10^4$. For our initial experiments we chose two very simple node loading and capacity scenarios (described in section 3 below). First we explain the structure of the simulation model.

Peersim divides time into cycles (we call these *Peersim Cycles*). Some number of Peersim cycles constitutes what we term a *Load Cycle*. In each Peersim cycle each node i is fired, in random order, and may receive a client query. If a query is received, it will be answered directly if the node is not currently overloaded. If the node is overloaded it will redirect the query to a randomly selected neighbor node if it has one. The neighbor node will then answer the query or, if it is also overloaded, will ignore it.

The number of queries (load) given to each node $1..N$ in one Load Cycle constitutes what we call the *Load Profile L*. For a node i, Li specifies the load for that node. If L specifies the same value for each node then all receive the same number of queries and their load is equal. Variation in loads between nodes indicates that some nodes receive more queries than others. Each node also stores an individual capacity value, Ci, that specifies the total number of queries it can serve over a Load Cycle. The set of all Ci capacity values for all nodes $1..N$ constitutes what we call the *Capacity Profile C*. Hence the sum of all Ci values constitutes the total system capacity, TC, which is the maximum number of queries the system as a whole can serve in one Load Cycle.

At the end of a Load Cycle each unsatisfied node con-

[1] http://peersim.sourceforge.net

siders moving in the overlay with probability $mp = 0.1$. A node is not satisfied if the proportion of queries, it directly received from clients, that were eventually served, ps, is below threshold t. Here we set $t = 1$ for all nodes, so a node is only satisfied if *all of its queries are served*. We currently do not model how nodes calculate the proportion of answered queries but assume some service or method exists to approximate it. Movement follows the scheme previously described.

We simulated two very simple query load scenarios on the nodes. In each case the node capacities were evenly distributed but the query loads were not. This means some nodes were overloaded and some under-loaded. In bother cases the Capacity and Load Profiles stay constant.

In the first scenario we set the Capacity Profile such that all nodes had capacity $C = 10$ queries and the Load Profile such that half the nodes had a load of 15 and the other half 5 queries. This means half the nodes are under-loaded by 5 and half are overloaded by 5. We set $k = 1$ for this scenario meaning that nodes only maintain a single link and therefore form pairings rather than complex network topologies.

In the second scenario we set the Capacity Profile such that all nodes had $C = 20$ and the Load Profile such that 10% of nodes had zero load, 10% of nodes had load 40, 40% of nodes had load 15 and the remaining 40% of nodes had load 25. This means that, like the first scenario, half the nodes are overloaded and half are under-loaded. But here nodes are over- or under-loaded by different amounts. This makes coordination a little more tricky than in the first scenario. We set $k = 4$ meaning that nodes maintain a maximum of four links to other nodes.

In all cases experimental results were produced by collecting results from 10 independent simulation runs that ran for 1000 load cycles each. To asses performance we recorded three measures, Q, S and M. Q gives the proportion of queries actually served as proportion of all submitted; S gives the proportion of nodes satisfied and M the amount of node movement as a proportion of the node population per load cycle. Q and S were calculated as averages over all runs for the last 500 cycles indicating performance after some level of stability is reached. M was calculated as an average over the entire 1000 load cycles indicating how much node movement (and hence replication effort) was required to reach and maintain the given performance.

4 Results

We present results from three different experiments (isolated, random and dynamic) for each of the two load scenarios. The *Isolated* experiments gives a performance baseline by running the simulation with all CacheWorld services turned-off: no links between nodes, no query redirection and no movement. This captures the situation where nodes simply answer their own queries and do not know about other nodes. The *Random* experiments give a secondary baseline. Here the overlay is initialized to a random topology (degree k) that is fixed - i.e. no node movement is possible. However, nodes my redirect their queries to neighbors if they become overloaded. In the *Dynamic* experiments the CacheWorld protocol is fully enabled allowing for dynamic node movement based on satisfaction, as previously described. By comparing *Random* and *Dynamic* experiments we can determine how much extra performance dynamic movement produces over just a fixed random overlay. By comparing *Isolated* and *Dynamic* experiments we can determine the overall increase in performance obtained by using CacheWorld over letting servers deal with all their own queries individually.

Results for scenario one are shown in figure 3. For the *Isolated* experiments we found, Q = 0.75, S = 0.5 with negligible variances. This is what we expected given the Load and Capacity Profiles. For the *Random* experiments we initialized the overlay network to a random topology with $k = 1$. This gives a further baseline showing how well a purely random approach performs. As might be expected there was some improvement because nodes have the chance to share load. We found $Q = 0.79$, $S = 0.58$. Finally in the *Dynamic* experiments the full CacheWorld protocol is used allowing movement by unsatisfied nodes. Again the initial topology of the overlay is randomly generated as before but nodes can change the topology over time by movement. We found improved performance with only 1% of queries not answered. We found $Q = 0.99$, $S = 0.98$, again variances were negligible. This improvement in performance comes at the cost of node movement (nodes making and breaking links to other nodes in the overlay). More movement occurs in the early Load Cycles and then the network settles to a more stable state. We found $M = 7.2 \times 10^{-4}$ which equates to a total of 36 node movements over 1000 cycles. This indicates that that a relatively small amount of movement suffices to improve performance. The main result here is that turning on the CacheWorld protocol improves total queries served, Q, by 24%, keeping 98% of nodes satisfied and resulting in only a small amount of node movement and hence replication effort. However, this is a very simple loading scenario in which load and capacities are constant and optimal pairs of nodes are easy to find so we would expect good results and indeed the scenario was selected as a minimal test.

For the second load scenario we conducted the same set of experiments. The results are shown in figure 3. Notice that for both the *Isolated* and *Random* experiments we gain comparable values for the number of satisfied nodes S but higher values for queries answered Q. This is due to the unbalanced nature of the load distributed and capacities which gives higher loads and capacities to some nodes. For

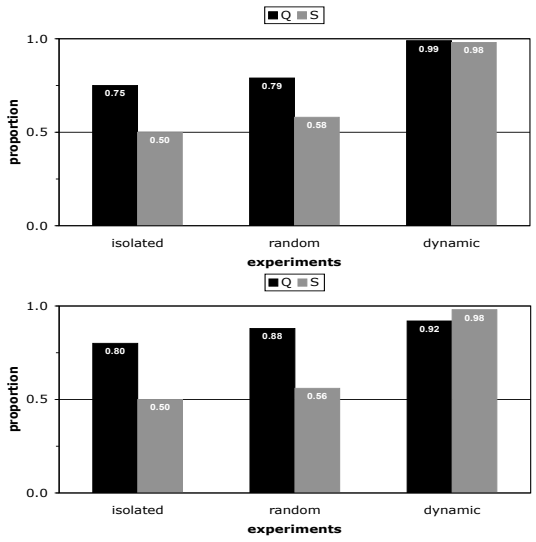

Figure 3. Results from three experiments for load scenario one (upper chart)) and two (lower chart). The scenarios are described in section 3 and experiments are described in section 4. Q = proportion of queries answered, S = proportion of nodes satisfied.

the *Dynamic* case we found $Q = 92$ and $S = 98$. However, more movement occurs compared to scenario one. We found $M = 9.5 \times 10^{-3}$ which equates to a total of 475 node movements over the 1000 cycles. This indicates that an order of magnitude more movement is required to obtain the performance increase in this more tricky, yet still relatively simple, scenario.

5 Discussion and future work

The idea behind CacheWorld is that the movement of nodes in the overlay network leads to the formation of clusters of servers that replicate each others content and deal with redirected queries. Elsewhere we have called similar formations $tribes$ [1]. A stable tribe is one that satisfies all of its members otherwise they will move away and over time the tribe will dissolve. Stable tribes comprise nodes with complementary loads (i.e. loads that tend to be negatively correlated). This may be plausible when demand for content follows daily patterns. Peers then would effectively supplement capacity for daytime load peaks in one time zone with spare overnight capacity from another.

In these initial experiments we have considered loads where an equal number of nodes are either under- or overloaded consistently so stable clusters can easily emerge. The results then, show, that at least in these simple scenarios our simple node movement heuristic is sufficient to produce stable clusters giving high performance. Although the scenarios are simple, we placed a 100% load on the system and set all node satisfaction thresholds to 100%. This means that the system only fully stabilizes when 100% of capacity is utilized.

A number of open issues need to be addressed. We have not tested the system under realistic query load (highly dynamic and skewed) and we have only modeled nodes holding a single monolithic content item (rather than many small items). We have not extensively tested the system with free-rider nodes that do not replicate, or serve queries, from others. However, initial experiments (not shown here) indicate that the system punishes free-riders because they need to move more often (twice as often as non-freeriders) because they tend to become isolated[2].

Several Replica Management systems already exist and are in use. However existing systems (e.g. Squirrel [3], Globule [4]), which manage replication, do not have specific mechanisms for adjusting the cooperation among nodes. For example, for balancing global policies (i.e. the goals of the community) against local (i.e. that of individual nodes / agents) policies. The policies in such systems are fixed by design. We aim to examine the applicability of augmenting these existing systems with our simple protocol to improve adaptability.

Our aim is to produce a distributed, scalable, robust and self-organising protocol. Our approach is not to guarantee optimality or absolute security but rather to produce very simple protocols utilising, where possible, existing technologies that perform reasonably well over a range of conditions.

References

[1] D. Hales and S. Arteconi. SLACER: A self-organizing protocol for coordination in peer-to- peer networks. *IEEE Intelligent Systems*, 21(2):29-35, Mar/Apr 2006.

[2] M. Jelasity, M. van Steen. Large-Scale Newscast Computing on the Internet. *Report IR-503, Vrije Universiteit Amsterdam, Dept. of Comp. Sci.*, Amsterdam, 2002.

[3] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: A decentralized peer-to-peer web cache. In *Proc. 21st ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC),* 2002.

[4] Guillaume Pierre and Maarten van Steen. Globule: a Collaborative Content Delivery Network *IEEE Communications Magazine* 44(8), pp. 127-133, August 2006.

[2] Free-riders were modeled as nodes that never answered other's queries and always redirected their queries, even when they had capacity.