

# Caching Strategies for a Distributed Swarm Computing Environment

**Sagnik Bhattacharya**  
*Department of Computer Science*  
*University of Virginia*  
*sagnik@virginia.edu*

## Abstract

In this paper we describe the theoretical framework for caching in a swarm network and propose a caching scheme which shall (a) reduce network traffic (b) reduces average response time, and (c) has low book-keeping overhead, all of which are essential for a swarm network based on wireless broadcast media.

## 1. Introduction

Swarm computing is a new computing paradigm enabled by the development of mass-producible micro-electro-mechanical devices (MEMs) with limited computing and (wireless) communication capabilities. Thousands of such devices, for example, can be sprayed around a volcano crater, submerged to the ocean floor, or injected into a human bloodstream. They would typically perform measurements and coordinate actions (such as emitting certain chemicals or even performing cooperative micro-surgery). These devices would typically run a suite of coordination protocols which control their communication, relay information to and from outside observers, interact with the physical environment, and provide suitable high-level abstractions to programmers and operators of the swarm. The collection of these protocols may define the next generation operating system for swarm computing.

There is a need for providing some sort of delay guarantees for requests/queries. There might be critical applications running on top of the swarm network, which shall fail if their deadlines are not met. For example, in a battlefield, a swarm of sensors can be

used to get useful information about the hostile environment. A tracking application which locks on to an enemy vehicle, needs to be able to get response to its queries about the vehicle's position at a certain minimum rate, failing which it shall lose track of its target.

Another application could be on the battlefields of the future. Marines will be able to check a handheld unit receiving data from a network of various tiny sensors to see and hear where the enemy is, how many there are, and even what sort of vehicle they are driving. Much of these shall be applications, which just make lazy periodic queries. So, it would be wasteful of network bandwidth to send these queries to the sensor every time. Some sort of caching mechanism could provide lower response time while reducing network traffic. Another side benefit of caching would be that sensors could turn themselves off as (standby mode), which would increase their battery life as well as prevent their detection.

We examine the use of network caching to aid in providing some sort of delay guarantees for critical applications. Our aim is to reduce response time, and reduce communication bandwidth.

## **2. Problem formulation**

### ***Task model:***

N nodes, M edge points ( $M < N$ ) and S sensors of K types.

Each edge point generates multiple periodic queries with period  $P_i$ .

The data size of each query is fixed ( $d_i$ ), and has it has an importance level ( $w_i$ ).

### ***Nodes :***

Each node has a network queue of size Q, and a cache size of C.

It has a location specified by  $(x_i, y_i)$  and a broadcast radius of R.

### ***Edge points :***

Generate periodic queries with period  $P_i$  .

### ***Sensors :***

Upon receiving a request from an edge point, sends the sensed data towards the edge point. The data has a time-to-live field associated with it, which specifies the

amount of time the data remains valid. Also, some events might occur at sensors, which causes it to invalidate its cached copies of data.

***Data model :***

All request/data packets are assumed to be of the same size. A packet is not addressed to a particular sensor or edge point. Instead, it is addressed by geographical location. A packet can be of K types corresponding to the K types of sensors. All packets have adequate headers attached to them and are of equal size.

***Network :***

The routing of packets is implemented as follows : Suppose a certain point the request packet for (x,y) is at node  $n_i$  . The node checks if it has a sensor in its neighborhood that is sufficiently close to the request position to serve the request. If so, then the request packet is passed onto that sensor. If there is no such sensor, then the request packet is passed onto the neighbor node which is nearest to the destination of the request. In case of data packets, they are served to the edge points instead of the sensors.

***Objective :***

1. To minimize the network traffic.
2. To reduce average response time, so as to minimize deadline misses.

### **3. Our Approach**

Initially, we played around with caching policies, and tried to gain an insight into how data distribution actually took place in the swarm network. The various parameters to be tuned were: cache size, time to live of the data, query generation rate, query generation pattern. The results are presented in the next section. Here we mainly worked with random query generation schemes. The cache strategies tried were :

1. Cache along the way : A data packet is cached at each hop if there is space at the cache. This works fine if the cache is not an issue, and the request generation rate is not too high, then this strategy works fine.
2. Cache along the way with replacement : Caching of a data packet is attempted at each hop. It is cached if there is space available, or if there is a packet with a lower

importance value. The new packet now replaces the packet with the least importance value

The results obtained for the given caching strategies are presented in section 5.

In the case that there is no constraints on the amount of cache space available at the intermediate nodes, we try to cache as near to the access points as possible. The caching protocol (strategy 3) used is as follows :

- When an edge point receives some data, it sends a rollback packet carrying the data back along the path through which it sent the request. The packet is cached at the first empty cache location along the way and sent no further.

This strategy gave much better results for network traffic generated, and performed well even when cache sizes were small. But, in this policy there is no prioritization of traffic. So, as a refinement to the above strategy (strategy 4), we proposed the following :

- When a node receives a rollback, it tries to seek a victim packet in its cache. The victim is the one which has the lowest importance value. In case of equal importance value, we select the packet which has less remaining life.

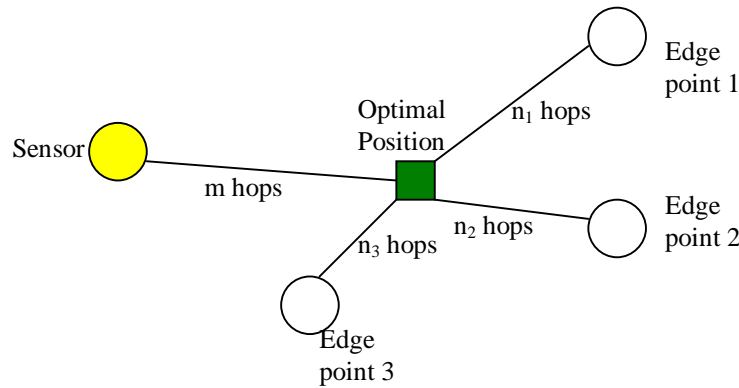
A basic drawback with all the schemes discussed above, is the development of an invalidation scheme. The direct approach would be to flood the network with invalidation packets. This would be very wasteful, and even more so given the knowledge that the packets are transmitted on wireless broadcast channels. Also, these schemes do not take into account the data dissemination patterns. For example, two edge points might be requesting the same sensor data, then it might be beneficial to serve them from the same cache copy, especially if there are constraints on cache space.

Now we can better identify the requirements for our caching policy. The points to consider are :

- Amount of network traffic for invalidation should be low.

- The amount of bookkeeping that can be done by a sensor to keep track of its data copies is low.
- While keeping ease of invalidation in mind, we also have to keep the data close enough to the edge points for the average response time to be low.

There is a need to build some theoretical foundations for our problem. Consider the following figure :



***Fig. 1***

Let  $R_{update}$  be the rate at which the data gets invalidated and  $R_1$ ,  $R_2$  and  $R_3$  be the rates at which edge points 1, 2 and 3 request data from the sensor. Let the optimal position  $(x,y)$  be such that it is  $m$  hops away from the sensor and  $n_1$ ,  $n_2$  and  $n_3$  hops away from edge points 1, 2 and 3 respectively. So, the net traffic rate is

$$TR = m.R_{update} + n_1.R_1 + n_2.R_2 + n_3.R_3.$$

such that,  $m + \min(n_1, n_2, n_3) \geq k$ , where  $k$  is some constant

If we generalize to  $N$  edge points,

$$TR = m.R_{update} + \sum_{1 \leq i \leq N} (n_i \cdot R_i)$$

such that,  $m + \min_{1 \leq i \leq N}(n_i) > k$ , where  $k$  is some constant.

Now, we want to minimize the traffic rate. So, we have at hand an optimization problem. We can consider our swarm network to be a continuum, and reduce our optimization problem to the following geometric problem.:

- Given  $N$  points, find a point  $(x,y)$  such that  $D = \sum_{1 \leq i \leq N} d_i \cdot w_i$  is minimum.  
 where,  $d_i$  is the distance of the  $i^{\text{th}}$  point from the point  $(x,y)$   
 and  $w_i$  is the edge weight of the edge from the  $i^{\text{th}}$  point to  $(x,y)$ .

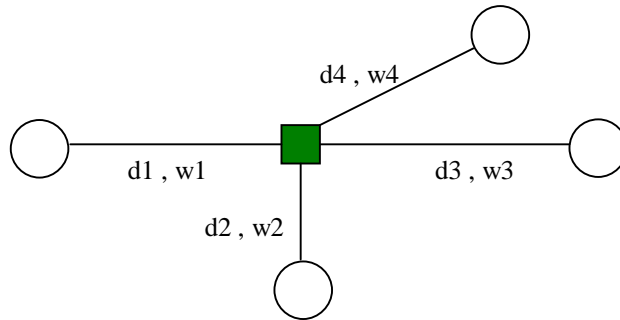


Fig. 2

In the given figure, we want to minimize  $D = d_1 \cdot w_1 + d_2 \cdot w_2 + d_3 \cdot w_3 + d_4 \cdot w_4$

We can then extend the optimization problem to have two copies of the data. So the corresponding geometrical problem would look like :

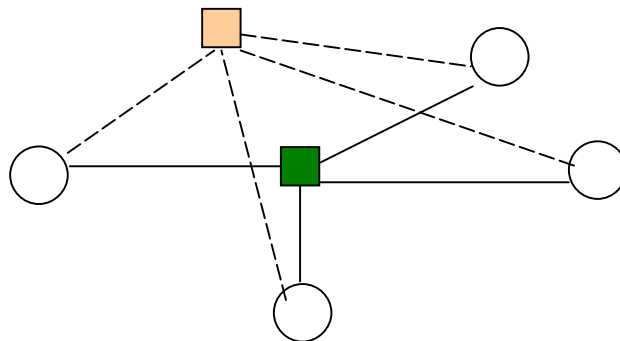


Fig. 3

Now we have two optimal locations  $(x,y)$  and  $(x',y')$ . So, we have to maximize :

$$D = d_1 \cdot w_1 + d_2 \cdot w_2 + d_3 \cdot w_3 + d_4 \cdot w_4 + d_1' \cdot w_1' + d_2' \cdot w_2' + d_3' \cdot w_3' + d_4' \cdot w_4'$$

where,  $d_i = 0$  if  $d_i' > 0$  and  $d_i' = 0$ , if  $d_i > 0$ .

Generalizing to N points :

$$D = \sum_{1 \leq i \leq N} d_i \cdot w_i + \sum_{1 \leq i \leq N} d_i' \cdot w_i' \text{ is the quantity we have to minimize.}$$

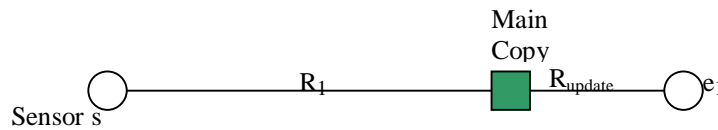
Extending this optimization problem further to M optimal points :

$$D = \sum_{1 \leq j \leq M} \sum_{1 \leq i \leq N} d_{ik} \cdot w_{ik} \text{ is the quantity we have to minimize.}$$

This optimization guarantees to give us a set of optimal points where to cache the data, so that the network traffic is minimal.

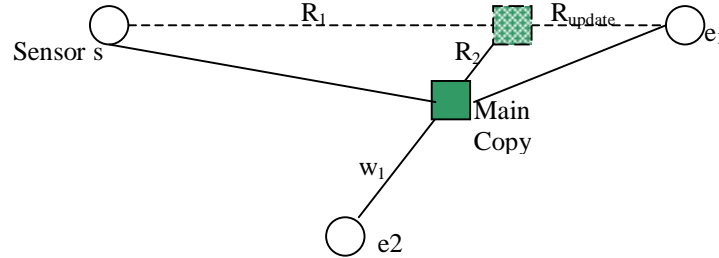
We now present a heuristic that tries to optimally place a data item in a sensor network, as well as to reduce response time. It works as follows :

- When an edge point,  $e_1$ , requests data from a sensor,  $s$ , the sensor sends the data and it is cached at a point  $c_1$ , which divides the path from  $e_1$  to  $s$ , in the ratio  $R_{update}:R_1$ . We call this copy the main copy and it is given a weight of  $w_1 = (R_{update}+R_1)$ . The sensor maintains a list of positions of main copies of the data. The edge point maintains a redirection table, which has mappings from sensor locations to main copy locations.



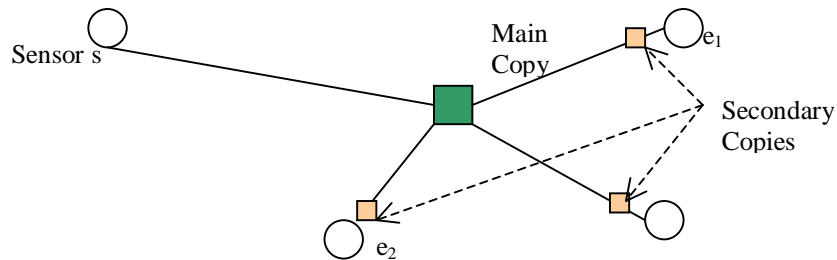
***Fig. 5***

- Suppose another edge point  $e_2$ , requests the same data. The sensor checks to see if it is closer to  $e_2$  than any of its main copies. If so, then it serves the request directly, and another main copy is created as in the previous step. Else, it determines the main copy which is nearest to  $e_2$  and sends a pointer to that main copy to  $e_2$ .
- Now, the main copy is moved to a position which divides the path from  $c_1$  to  $e_2$  in the ratio  $R_2:w_1$ . The weight  $w_1$  is now incremented by  $R_2$ . The main copy list of the sensor is updated and also the mapping tables of all the edge points requesting the sensor data are updated.



***Fig. 6***

- Similarly if any other edge point requests the sensor data , then it is served similarly, and the main copy gets shifted.
- Another level of cache hierarchy is added. Each edge point is allowed to have secondary copies of the main copy, with the constraint that the time-to-live for the secondary copy is  $k$ -times that of the main copy, (where  $k \ll 1$ ). The creation and replacement of secondary cache copies are done using the caching strategy 4, that was discussed earlier in this paper. The main copies are however locked into cache.



***Fig. 7***

The advantages of this caching heuristic are as follows :

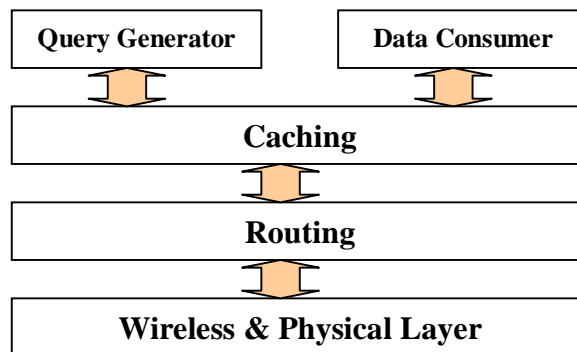
- This approaches the optimal placement (intuitively, no formal proof available yet), for the main copies. Thus network traffic is minimized.
- By introducing secondary caching, we further reduce average response time, though running the risk of getting stale data on rare occasions.

- Amount of bookkeeping for invalidation of cache copies is ver small. The sensor only has a finite number of main copies. For invalidation, it only needs to send invalidation messages to these copies. The secondary copies have a very small time-to-live, so they invalidate themselves.

#### 4. Implementation

Various caching strategies were tested using a scalable swarm network simulator. The main features of the simulator are as follows :

- The simulator randomly places a specified number of nodes, sensors and edge points on a 100x100 grid.
- Each node builds up a neighborhood table, which contains all the nodes within broadcast radius of the node. This might not actually happen in the physical environment, but helps in the simulation of the wireless layer, and makes the computation fast.
- The layered structure of the simulator can be summarized as follows :



*Fig. 8*

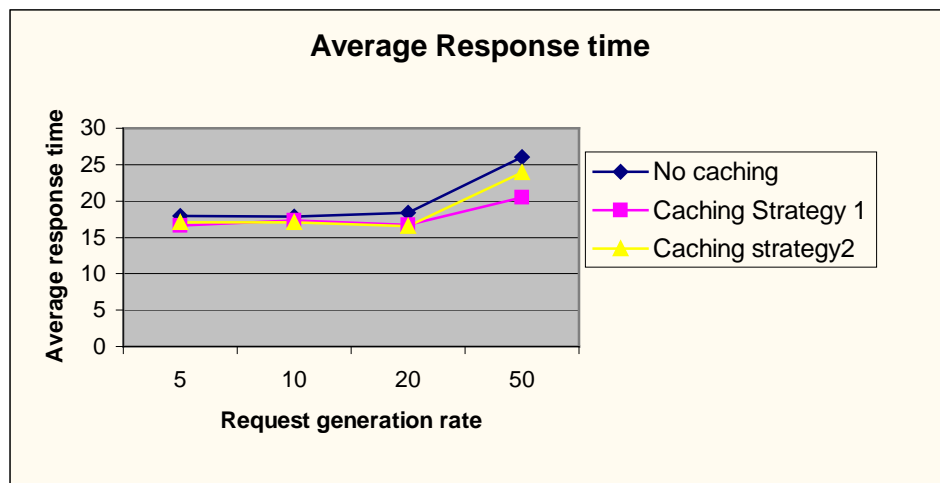
- The simulator is built such that any caching module can be inserted at the caching layer. It only has to implement the interface

```
public interface Cache {
    public boolean shouldCache(Node n, Packet p, int time);
    public void putInCache(Node n, Packet p, int time);
}
```

- Routing strategies 1, 2, 3 and 4 were implemented and the results are displayed in the next section.

## 5. Performance

The graph for average response time versus request generation rate for random load is given below for the cases when no caching, caching strategy 1 and caching strategy 2 were used.

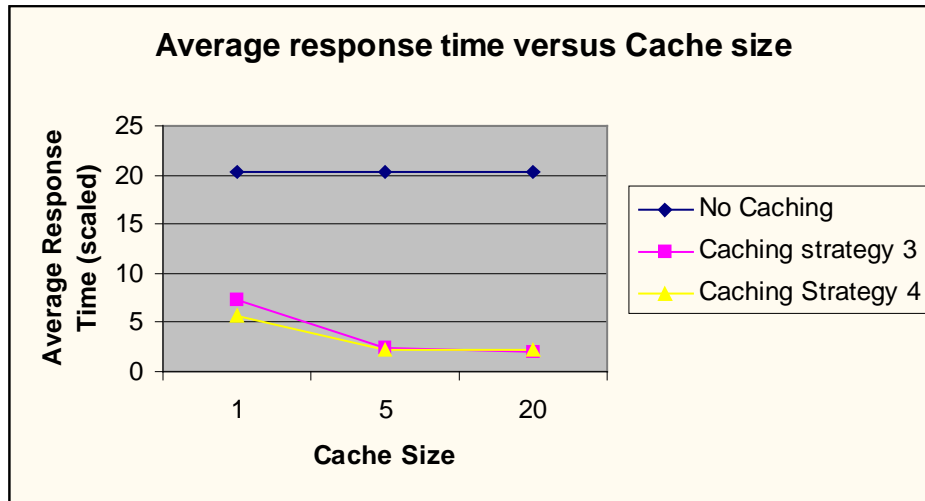


*Fig. 9*

As we can see, even for a random request load, caching leads to an improvement in average response time, more so as the request generation rate increases. However, we can see that as request generation rate increases, the response time for strategy 2 (basically strategy 1 with replacement), increases because of thrashing. Cached pages are replaced before they can actually serve requests.

So, now we concentrate on periodic concentrated queries. We also refine our metrics, so now instead of just having average response time, we have average response time which is scaled by the importance value, i.e., an item with greater importance value has greater impact on average response time.

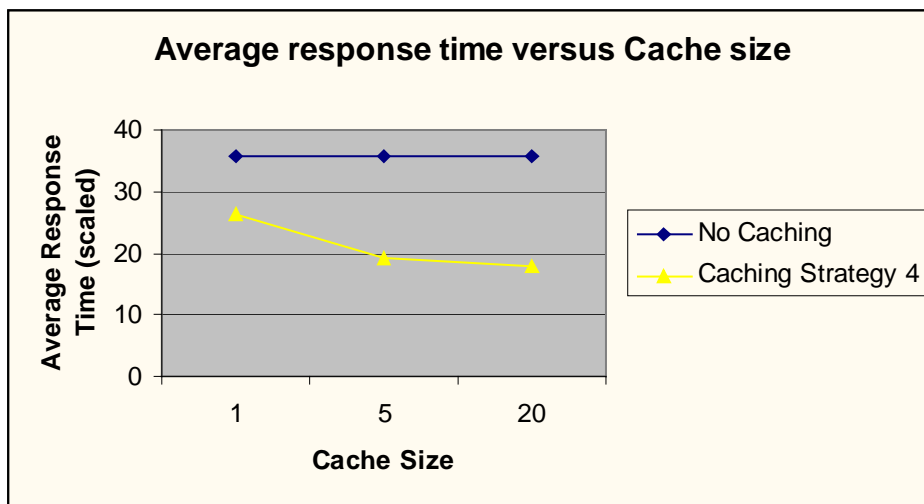
The following graph plots the average response time( scaled) against the cache size for caching strategies 3 and 4.



*Fig. 10*

As we can see, caching strategy 4 gives pretty satisfactory results, and it is easily apparent that it is the optimal strategy for in a case when there is only one edge point. Thus, we use this to perform secondary caching from the main copies in our caching scheme.

The following graph shows that the scheme still gives good results even in the face of convergent queries, i.e., many edge points ask for the same data. The following graph demonstrates it.



*Fig. 11*

## 6. Related work

Swarm networks have not been extensively studied in networking literature. Especially no work seems to have been done on caching as a means of providing guarantees in swarm networks. Some groups are working on sensor networks, but their work focuses on different aspects of the swarm computing environment.

The SCADDS group at UCLA is working on a routing paradigm in sensor networks called directed diffusion, which borrows heavily from ad-hoc unicast routing. But the overhead for their strategies are much more compared to our work, and they assume more computational ability for their sensors than we do. Also, their testing bed has about 50 to 250 nodes while we have tested with around 10,000 nodes. But, they have also considered power consumption, which have not really taken into consideration.

The SenseIT project sponsored by DARPA is another effort, which involves a large number of research groups, but not much information is available as to their activities.

Some sort of parallel can be drawn with distributed caching on the web, more accurately caching at routers, but not much headway has been made into that due to protocol issues. However, work has been done on adaptive distributed web caching using co-operative proxy caches.

## 7. Conclusion and Future Work

In our paper, we described our experience with various caching heuristics for a swarm network, and we came up with a heuristic, which we claim shall give better performance by:

- Reducing network traffic.
- Reducing request response time.
- Low overhead for the nodes.

The last point is very relevant because a swarm network consists of micro-devices, with limited computational power and memory. It might not be possible to overload them with a heavy protocol stack, or for them to run complex optimization algorithms. Our heuristic makes it simple for nodes to provide dependable caching with very little overhead.

Unfortunately the heuristic hasn't been tested yet. Some more theoretical work needs to be done on that. But, it can be said with conviction that it shall perform better than the heuristics described above, as it combines a distributed optimization heuristic and an optimal caching algorithm (strategy 5).

Another aspect that needs to be looked at is the effect of caching on energy consumption. Sensors can go to power-save mode while the requests are being served from cache. This can be invaluable, because the battery life of a sensor determines its lifespan. One cannot expect to recharge a sensor in a battlefield!

There may be a lot of other issues, which though not so apparent, are important all the same. A lot of work still needs to be done in this field. To quote [1] : we are with sensor networks where we were with the Internet 3 decades ago.

## **8. References**

- (1) *The MURI proposal* - John Stankovic, Tarek Abdelzaher.
- (2) "DARPA technology visits MCAGCC, tests new technology for field use" - LCpl. Brent Walker. [http://www.29palms.usmc.mil/base/PAO/23mar\\_st1.html](http://www.29palms.usmc.mil/base/PAO/23mar_st1.html)
- (3) "Directed Diffusion: A Scalable and Robust Communication paradigm for Sensor Networks." - C. Intanagonwiwat, R. Govindan, D. Estrin
- (4) "Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks" - Ya Xu, J. Heidemann, D. Estrin
- (5) "Smart Dust: Communicating with a cubic Millimeter Computer" - B. Warneke, M. Last, B. Liebowitz, K.S.J. Pister
- (6) "Amorphous Computing" - Abelson, Allen, Coore, Hanson, Rauch, Sussman, Weiss.
- (7) "Design considerations for distributed caching on the internet." - R.Tewari, M. Dahlin, H.M.Vin, J.S.Kay.