On Deploying Tree Structured Agent Applications in Networked Embedded Systems

Nikos Tziritas[1,3], Thanasis Loukopoulos[2,3], Spyros Lalis[1,3], and Petros Lampsas[2,3]

[1] Dept. of Computer and Communication Engineering, Univ. of Thessaly, Glavani 37, 38221 Volos, Greece
{nitzirit,lalis}@inf.uth.gr
[2] Dept. of Informatics and Computer Technology, Technological Educational Institute (TEI) of Lamia, 3rd km. Old Ntl. Road Athens, 35100 Lamia, Greece
{luke,plam}@teilam.gr
[3] Center for Research and Technology Thessaly (CERETETH), Volos, Greece

Abstract. Given an application structured as a set of communicating mobile agents and a set of wireless nodes with sensing/actuating capabilities and agent hosting capacity constraints, the problem of deploying the application consists of placing all the agents on appropriate nodes without violating the constraints. This paper describes distributed algorithms that perform agent migrations until a "good" mapping is reached, the optimization target being the communication cost due to agent-level message exchanges. All algorithms are evaluated using simulation experiments and the most promising approaches are identified.

1 Introduction

Mobile code technologies for networked embedded systems, like Aggila [1], Smart-Messages [2], Rovers [3] and POBICOS [4], allow the programmer to structure an application as a set of mobile components that can be placed on different nodes based on their computing resources and sensing/actuating capabilities. From a system perspective, the challenge is to optimize such a placement taking into account the message traffic between application components.

This paper presents distributed algorithms for the dynamic migration of mobile components, referred to as agents, in a system of networked nodes with the objective of reducing the network load due to agent-level communication. The proposed algorithms are simple so they can be implemented on nodes with limited memory and computing capacity. Also, modest assumptions are made regarding the knowledge of routing paths used for message transport. The algorithms rely on information that can be provided by even simple networking or middleware logic without incurring (significant) additional communication overhead.

The contributions of the paper are the following: (i) we identify and formulate the agent placement problem (APP) in a way that is of practical use to the POBICOS middleware but can also prove useful to other work on mobile agent systems with placement constraints, (ii) we present a distributed algorithm that relies on minimal network knowledge and extend it so that it can exploit additional information about

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part II, LNCS 6272, pp. 490–502, 2010.

the underlying network topology (if available), (iii) we evaluate both algorithm variants via simulations and discuss their performance.

2 Application and System Model, Problem Formulation

This section introduces the type of applications targeted in this work and the underlying system and network model. It then formulates the agent placement problem (APP) and the respective optimization objectives.

2.1 Application Model

We focus on applications that are structured as a set of cooperating agents organized in a hierarchy. For instance, consider a demand-response client which tries to reduce power consumption upon request of the energy utility. A simplified possible structure is shown in Fig. 1. The lowest level of the tree comprises agents that periodically report individual device status and power consumption to a room agent, which reports (aggregated) data for the entire room to the root agent. When the root decides to lower power consumption (responding to a request issued by the electric utility), it requests some or all room agents to curve power consumption as needed. In turn, room agents trigger the respective actions (turn off devices, lower consumption level) in the end devices by sending requests to the corresponding device agents.

Leaf (sensing and actuating) agents interact with the physical environment and must be placed on nodes that provide the respective sensors or actuators and are located at the required areas, hence are called *node-specific*. On the other hand, intermediate agents perform their tasks using just general-purpose computing resources which can be provided by any node; thus we refer to these agents as *node-neutral*. In Fig. 1, device agents are node-specific while all other agents are node-neutral.

Agents can migrate between nodes to offload their current hosts or to get closer to the agents they communicate with. In our work we consider migration *only* for node-neutral agents because their operation is location- and node-independent by design, while node-specific agents remain fixed on the nodes where they were created. Still, the ability to migrate node-neutral agents creates a significant optimization potential in terms of reducing the overall communication cost.

2.2 System Model

We assume a network of capacitated (resource-constrained) nodes with sensing and/or actuating capabilities. Let n_i denote the i^{th} node, $1 \leq i \leq N$ and $r(n_\mathrm{i})$ its resource capacity (processing power or memory size). The capacity of a node imposes a generic constraint to the number of agents it can host.

Nodes communicate with each other on top of a (wireless) network that is treated as a black box. The underlying routing topology is abstracted as a graph, its vertices representing nodes and each edge representing a bidirectional routing-level link between a node pair. In this work we consider *tree-based routing*, i.e., there is exactly one path for connecting any two nodes. Let D be a $N \times N \times N$ boolean matrix encoding the routing topology as follows: $D_{\mathrm{ijx}}=1$ iff the path from n_i to n_j includes n_x, else 0. Since we assume that the network is a tree $D_{\mathrm{ijx}} = D_{\mathrm{jix}}$. Also, $D_{\mathrm{iii}}=1$, $D_{\mathrm{ijj}}=1$ and $D_{\mathrm{iij}}=0$. Let h_{ij} be the path length between n_i and n_j; equal to 0 for $i=j$. Obviously, $h_{ij} = h_{ji}$.

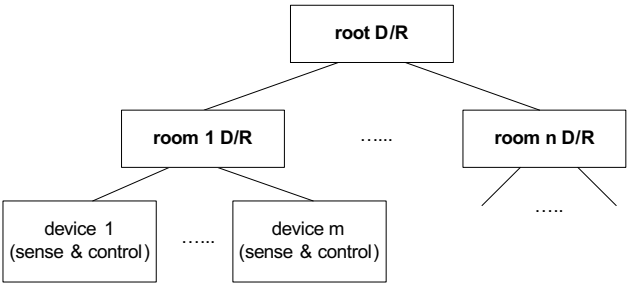

Fig. 1. Agent tree structure of an indicative sensing/control application

Each application is structured as a set of cooperating agents organized in a tree, the leaf agents being node-specific and all other agents being node-neutral. Assuming an enumeration of agents whereby node-neutral agents come first, let a_k be the k^{th} agent, $1 \leq k \leq A+S$, with A and S being equal to the total number of node-neutral and node-specific agents, respectively. Let $r(a_k)$ be the capacity required to host a_k. Agent-level traffic is captured via an $(A+S)\times(A+S)$ matrix C, where C_{km} denotes the load from a_k to a_m (measured in data units over a time period). Note that C_{km} need not be equal to C_{mk}. Also, $C_{kk}=0$ since an agent does not send messages to itself.

2.3 Problem Formulation

For the sake of generality we target the case where all agents are already hosted on some nodes, but the current placement is non-optimal.

Let P be an $N\times(A+S)$ matrix used to encode the placement of agents on nodes as follows: $P_{ik}=1$ iff n_i hosts a_k, 0 otherwise. The total network load L incurred by the application for a placement P can then be expressed as:

$$L = \sum_{k=1}^{A+S} \sum_{m=1}^{A+S} C_{km} \sum_{i=1}^{N} \sum_{j=1}^{N} h_{ij} P_{ik} P_{jm} \tag{1}$$

A placement P is valid iff each agent is hosted on exactly one node and the node capacity constraints are not violated:

$$\sum_{i=1}^{N} P_{ik} = 1, \ \forall 1 \leq k \leq A+S \tag{2}$$

$$\sum_{k=1}^{A+S} P_{ik} r(a_k) \leq r(n_i), \ \forall 1 \leq i \leq N \tag{3}$$

Also, a migration is valid only if starting from a valid placement P it leads to another valid agent placement P' without moving any node-specific agents:

$$P_{ik} = P'_{ik}, \ \forall A < k \leq A+S \tag{4}$$

The agent placement problem (APP) can then be stated as: starting from an initial valid agent placement P^{old}, perform a series of valid agent migrations, eventually leading to a new valid placement P^{new} that minimizes (1).

Note that the solution to APP may be a placement that is actually suboptimal in terms of (1). This is the case when the optimal placement is infeasible due to (3), more specifically, when it can be reached only by performing a "swap" that cannot be accomplished because there is not enough free capacity on any node. A similar feasibility issue is discussed in [5] but in a slightly different context. Also, (1) does not take into account the cost for performing a migration. This is because we target scenarios where the application structure, agent-level traffic pattern and underlying routing topology are expected to be sufficiently stable to amortize the migration costs.

3 Uncapacitated 1-Hop Agent Migration Algorithm

This section presents an agent migration algorithm for the case where nodes can host any number of agents, i.e., without taking into account capacity limitations. In terms of routing knowledge, each node knows only its immediate (1-hop) neighbors involved in transporting inbound and outbound agent messages; we refer to this as *1-hop network awareness*. This information can be provided by even a very simple networking layer. A node does not attempt to discover additional nodes but simply considers migrating agents to one of its neighbors. An agent may nevertheless move to distant nodes via consecutive 1-hop migrations.

Description. The *1-hop agent migration algorithm* (AMA-1) works as follows. A node records, for each locally hosted agent, the traffic associated with each neighboring node as well as the local traffic, due to the message exchange with remote and local agents, respectively. Periodically, this information is used to decide if it is beneficial for the agent to migrate to a neighbor.

More formally, let l_{ijk} denote the load associated with agent a_k hosted at node n_i for a neighbor node n_j (notice that n_i does not have to know D to compute l_{ijk}):

$$l_{ijk} = \sum_{m=1}^{A+S} (C_{km} + C_{mk}) D_{ixj}, P_{xm} = 1 \tag{5}$$

The decision to migrate a_k from n_i to n_j is taken iff l_{ijk} is greater than the total load with all other neighbors of n_i plus the local load associated with a_k:

$$l_{ijk} > l_{iik} + \sum_{x \neq i,j} l_{ixk}, \quad h_{ij} = h_{ix} = 1 \tag{6}$$

The intuition behind (6) is that by moving a_k from its current host n_i to a neighbor n_j, the distance for the load with n_j decreases by one hop while the distance for all other loads, including the load that used to take place locally, increases by one hop. If (6) holds, the cost-benefit of the migration is positive, hence the migration reduces the total network load as per (1).

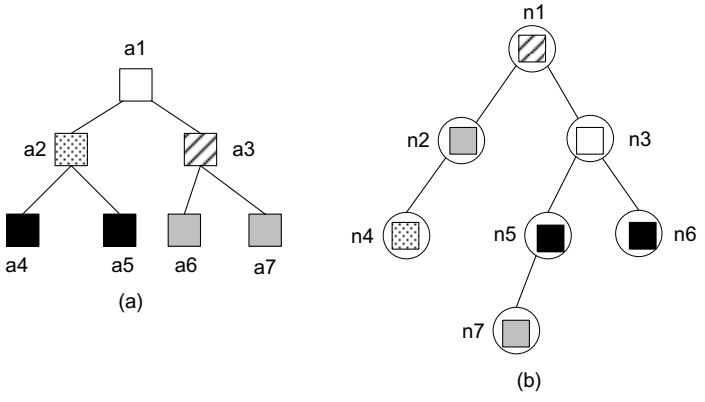

Fig. 2. (a) Application agent structure; (b) Agent placement on the network

Consider the application depicted in Fig. 2a which comprises four node-specific agents (a_4, a_5, a_6, a_7), two intermediate node-neutral agents (a_{42} a_3) and a node-neutral root agent (a_1), and the actual agent placement on nodes shown in Fig. 2b. Let each node-specific agent generate 2 data units per time unit towards its parent, which in turn generates 1 data unit per time unit towards the root (edge values in Fig. 2a). Assume that n_1 runs the algorithm for a_3 (striped). The load associated with a_3 for the neighbor node n_2 and n_3 is l_{123}=2 respectively l_{133}=3 while the local load is l_{113}=0. According to (6) the only beneficial migration for a_3 is for it to move on n_3. Continuing the example, assume that a_3 indeed migrates to n_3 and is (again) checked for migration. This time the relevant loads are l_{313}=2, l_{353}=2, l_{363}=0, l_{333}=1, thus a_3 will remain at n_3. Similarly, a_1 will remain at n_3 while a_2 will eventually migrate from n_4 to n_2 then to n_1 and last to n_3, resulting in a placement where all node-neutral agents are hosted at n_3. This placement is stable since there is no beneficial migration as per (6).

Implementation and complexity. For each local agent it is required to record the load with each neighboring node and the load with other locally hosted agents. This can be done using a $A' \times (g+1)$ load table, where A' is the number of local node-neutral agents and g is the node degree (number of neighbors). The destination for each agent can then be determined as per (6) in a single pass across the respective row of the load table, in $O(g)$ operations or a total of $O(gA')$ for all agents. Note that the results of this calculation remain valid as long as the underlying network topology, application structure and agent message traffic statistics do not change.

Convergence. The algorithm does *not* guarantee convergence because it is susceptible to livelocks. Revisiting the previous example, assume that the application consists only of the right-hand subtree of Fig. 2a, placed as in Fig. 2b. Node n_1 may decide to move a_3 to n_3 while n_3 may decide to move a_1 to n_1. Later on, the same migrations may be performed in the reverse direction, resulting in the old placement etc.

We expect such livelocks to be rare in practice, especially if neighboring nodes invoke the algorithm at different intervals. Nevertheless, to guarantee convergence we introduce a coordination scheme in the spirit of a mutual exclusion protocol. When n_i decides to migrate a_k to n_j it asks for a permission. To avoid "swaps" n_j denies this

request if: (i) it hosts an agent $a_{k'}$ that is the child or the parent of a_k, (ii) it has decided to migrate $a_{k'}$ to n_i, and (iii) the identifier of n_j is smaller than that of n_i ($j<i$). Else, n_j grants permission to n_i and does not consider migrating any child or parent of a_k to n_i before the granted migration completes. Convergence is guaranteed since it is no more possible to perform swaps and each migration that is not a swap reduces the network load as per (1). It is worth pointing out that such a protocol can be implemented quite efficiently by piggybacking requests and replies on other messages that need to be exchanged anyway in order to perform the actual migration.

4 Uncapacitated *k*-Hop Agent Migration Algorithm

This section introduces an extension of the 1-hop algorithm for the case where a node is assumed to know the routing topology within a k-hop radius. We refer to this as *k-hop network awareness*. Note this information may be collected in a lazy fashion, incurring a minimal communication overhead, by piggybacking the k most recent node identifiers when a (small) message travels through the network. In fact, this information comes for free by employing a naming scheme that encodes path information into node identifiers (e.g., as in ZigBee networks with hierarchical routing).

Description. The *k-hop agent migration algorithm* (AMA-*k*) is a straightforward extension of AMA-1 that exploits k-hop awareness. The difference is that for each agent a_m hosted at node n_i, AMA-*k* considers as possible candidates all nodes up to *k-hops* away from n_i which are involved in the message traffic of a_m.

The algorithm chooses the destination for a_m by iteratively evaluating (6) for neighbor nodes, starting from 1-hop neighbors and working its way to more distant neighbors, following the most beneficial outbound direction. Each iteration determines whether it is beneficial to move a_m to a node that is 1 hop further away from n_i assuming a_m were hosted on the node picked in the previous iteration. The algorithm stops after k iterations or earlier when it is no longer beneficial to migrate a_m. AMA-*k* is expected to lead to fewer migrations than AMA-1 because an agent can (directly) move on a distant node in a single migration; as opposed to performing several 1-hop migrations to reach the same destination.

Returning to the previous example of Fig. 2, assume that node n_4 runs AMA-5 for agent a_2. The first iteration will determine that a_2 should migrate (from n_4) to n_2, the second iteration will determine that a_2 should migrate (from n_2) to n_1, the third iteration will determine that a_2 should move on n_3, and finally the fourth iteration will decide that it is not beneficial for a_2 to migrate any further. At this point the algorithm stops, suggesting the migration of a_2 from n_4 to n_3.

Implementation and complexity. AMA-*k* requires the same type of load information as AMA-1 but for all k-hop instead of just 1-hop neighbors, yielding g^k the space complexity of AMA-1 (note that a refined, asynchronous, implementation, could store only the loads of the neighbors that are relevant for the computation o each iteration, requiring the same amount of memory as AMA-1). The destination for an agent is chosen in up to k iterations, each time evaluating (6) for the relevant, up to g, neighbor nodes, yielding a total time complexity of $O(kg)$ for determining the most beneficial destination for a local agent, i.e., AMA-*k* is k times slower than AMA-1.

Convergence. It can be shown that the algorithm converges provided that race conditions are tackled as per AMA-1.

5 Handling Capacity Constraints

This section discusses how AMA-1 and AMA-*k* can be extended to handle node capacity constraints. When running the algorithms, some assumptions must be made regarding the free capacity of remote nodes to drop infeasible solutions. Notably, these assumptions could be invalid and must be confirmed in order to actually perform a migration. In this paper we investigate two different schemes, as follows.

Inquire-Lock Before (ILB). Before running the algorithm, a request is sent to *all potential destinations*, 1-hop or *k*-hop neighbors depending on the algorithm, inquiring about their free capacity and requesting to reserve up to the amount needed to host *all* locally hosted agents that could be selected for migration. Nodes reply with their available free capacity, if any, which they reserve until further notice. The selection of the destination for each locally hosted agent is done as described in the previous subsections, having a *consistent and guaranteed view* of node capacities. When the destinations are chosen, all other nodes are informed to release the reserved capacity, while destinations release the capacity that is left over after accepting the agents assigned to them.

Inquire-Lock After (ILA). The algorithm runs based on a previous, *possibly outdated*, view of free node capacities. Destinations are then contacted to reserve the capacity needed for hosting the agents assigned to them. Initially, all nodes are assumed to have an infinite free capacity. This view, along with the nominal capacity of each node, is updated based on the replies received for each request. To avoid excluding destinations due to outdated information, with a certain probability nodes are assumed to have their full nominal capacity free, independently of the local view. Of course, this means that a migration might be decided based on invalid information, in which case the destination will send a negative reply when contacted to actually reserve capacity (and perform the migration).

Algorithmic adaptations. When AMA-1 picks a destination for a locally hosted agent, the migration is performed only if that node indeed has sufficient free capacity. Else, the agent is not considered for migration because all other destinations are guaranteed to lead to a load increase; (6) holds for at most one 1-hop neighbor or put in other words there can be at most one beneficial migration direction in a tree network. In contrast, AMA-*k* can fall back to the next best option in that path. For instance, in Fig. 2, n_4 would consider first n_3, then n_1 and finally n_2 as destinations for the migration of a_2. Notably, the destinations chosen by ILB are guaranteed to be able to host the agents assigned to them, while ILA may pick destinations that turn out not to have sufficient free capacity to host the agent(s) assigned to them.

Starvation. It is important to note that both schemes are subject to *starvation* due to races. More specifically, if A and B attempt to reserve capacity Ra and Rb (Ra<Rb) from C (with free capacity Ra<Rc<Rb), both A and B may fail (if B's request arrives

to C before A's request) even though at least one could have succeeded (C has enough capacity to satisfy A's request). To reduce the probability of such races, nodes can invoke the algorithm at the same period but with a random offset/delay. As an additional precaution, a back-off scheme can be applied in case a reservation attempt fails. Of course, this does not guarantee convergence.

6 Experiments

This section presents an experimental evaluation of the algorithms based on simulations performed on top of NS2. First we describe the experimental setup and then we present and discuss the results of indicative experiments.

6.1 Setup

Two types of networks are considered with 20 and 50 nodes placed randomly in a 80×80 and 120×120 plane, respectively. Nodes are in range of each other if their Euclidean distance is less than 30. The tree-based routing topology is obtained by calculating a spanning tree over the connectivity graph. Five topologies are generated for each network type. Each experiment is performed on all topologies. The average diameter for the 20- and 50-node networks is 6 and 15, respectively.

The application structure is generated as follows. Starting from an initial set of node-specific (leaf) agents, agents are split in disjoint groups of 5, and for each group 2-5 agents are randomly chosen, removed from the set, and labeled as children of a new node-neutral agent that is added to the set. This process is repeated until the set comprises a single agent which becomes the root (we check to make sure that this is indeed a node-neutral agent). Three application structures are generated with (50, 22), (25, 12) and (10, 5) (node-specific, node-neutral) agents, referred to as app50, app25 and app10, respectively. The initial agent placement on nodes is random.

In terms of application-level traffic, we let each node-specific (leaf) agent send 10-50 messages per time unit to its parent and each node-neutral (intermediate) agent send to its parent the average of the load received from its children (perfect aggregation). Also, each parent agent sends 1 message per time unit to its children (representing a heartbeat protocol). For simplicity, all messages are of equal size. The traffic pattern is stable throughout the whole duration of the experiments.

Nodes invoke the algorithm every T time units. Each node starts its periodic invocation with a different offset, randomly set between 0 and T. If an attempted migration fails due to resource constraints, the node backs-off for a number of periods T, chosen randomly between 1 and 5. Finally, in ILA, the probability for considering a node assuming that its full nominal capacity is free (as opposed to its free capacity according to the local view) is set to 20%.

As the main metric for our comparison, we measure the network load that corresponds to the agent placement produced by the algorithms vs. the load of the initial random placement but also vs. the optimal solution obtained via an exhaustive search algorithm (only for small-scale experiments). For experiments without capacity constraints, convergence is inferred when all nodes invoke the algorithm without attempting any migration. In experiments with capacity constraints, where algorithms employ

the ILB or ILA scheme and convergence is not guaranteed, the simulation is stopped when each node invokes the algorithm 4 consecutive times without managing to perform a migration. The overhead of algorithms is captured via the number of agent migrations performed to reach the final placement as well as the number of (control) messages exchanged to avoid swaps and to reserve and release capacity.

6.2 Results without Capacity Constraints

In a first experiment we compare the placements obtained by the uncapacitated algorithms for the 20-node networks and one app10 application. Table 1 summarizes the results for different degrees of network awareness (average values for the 5 different topologies). All algorithms perform close to optimal, even though the initial random placement is very bad, incurring more than twice the load of the optimal solution.

Table 1. Performance in the uncapacitated case (20 nodes, app10)

Algorithm	Total Load	Migrations	Control Msgs
Initial	106,6	-	-
AMA-1	45	10	20
AMA-2	44,4	6,8	13,6
AMA-3	44,8	5,2	10,4
AMA-4	44,8	5	10
Optimal	43,6	-	-

The (slightly) inferior placement achieved by AMA-1 is due to the fact that it forces distant migrations to occur in iterations, moving agents one hop at a time. In the meantime, other agents that communicate with the agent "under migration" might migrate too, leading to a suboptimal lock-in. Greater network awareness reduces the probability of such lock-ins but does not guarantee their absence, e.g., note that AMA-3 and AMA-4 produce a (slightly) worse placement than AMA-2.

As expected, greater network awareness leads to fewer migrations because agents can be placed directly on nodes further away from their original hosts, if desired. Notice that the number of control messages (in this case generated to avoid swaps) equals twice the number of migrations, indicating that no migration was turned down.

6.3 Results with Capacity Constraints – Small Scale Experiments

In a second experiment, for capacitated nodes, we compare AMA-1 and AMA-2 vs. the optimal solution for the same topology and application as before, for both ILB and ILA schemes. All agents have identical capacity requirements. The results are plotted as node capacity is increased so that each node can host 1, 2, 3 and 4 additional agents compared to the initial placement.

As it can be seen in Fig. 3, all algorithms produce sub-optimal results when node capacity is scarce, but the gap shrinks quite rapidly as capacity becomes abundant, approaching the results of the exhaustive search algorithm. Once again, the placements achieved by AMA-2 are better than those of AMA-1. Somewhat surprisingly, ILB consistently outperforms ILA only for AMA-1 but not for AMA-2. When capacity is

tight, AMA-2 produces better results with ILA than ILB, even though ILA works with possibly outdated node capacity information. This can be explained due to the greedy locking approach of ILB which leads to more collisions compared to ILA, as network awareness increases and a node can receive capacity reservation requests from a larger number of nodes. Also, the probability of ILA missing out on beneficial migrations is smaller when node capacity is scarce (most likely it is used anyway).

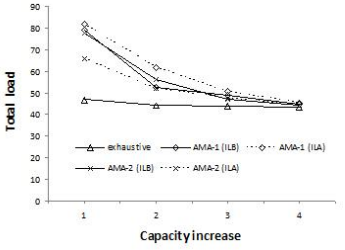

Fig. 3. Total load vs. capacity increase (20 nodes, app10)

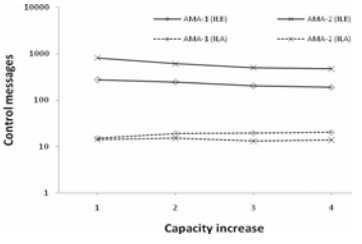

Fig. 4. Control overhead vs. capacity increase (20 nodes, app10)

Another negative effect of ILB is shown in Fig. 4 which plots the number of generated control messages. ILB clearly incurs a significantly higher overhead compared to ILA, by 1.5–2 orders of magnitude. This is due to the fact that ILB pro-actively inquires about and attempts to reserve free capacity on *all* neighbor nodes within a *k*-hop radius, while ILA mainly relies on information acquired through previous communications and tries to lock *only* the nodes that are actually selected as destinations.

6.4 Results with Capacity Constraints – Large Scale Experiments

We also performed experiments for the 50-node networks and an application mix of five instances of app10, app25 and app50. We compare the performance of AMA-*k*, for *k*=1, 2, 5, 10. Given the bad scalability of ILB, obvious from the previous results, only ILA is used. In the spirit of the previous experiments, the algorithms were tested for the case where each node is capable of hosting 5, 10, 20 and 40 additional agents compared to the initial random placement.

Fig. 5 and Fig. 6 depict the load corresponding to the placements achieved (the initial placements amounted to an average load of 11,000) and the number of migrations performed to reach them, respectively. As expected, greater network awareness results in better placements and fewer migrations. The differences in placement quality are more pronounced for limited capacity and shrink as capacity increases, while the opposite trend holds for the number of migrations. Note that capacity constraints have a greater impact for smaller values of *k*. This is because, as discussed in Sec. 6.2, low network awareness is more likely to lead to suboptimal lock-ins, but now this may also waist capacity that could have enabled more beneficial migrations. Indeed this effect is more visible when capacity is scarce and diminishes as capacity increases.

The number of control messages is plotted in Fig. 7. AMA-1 and AMA-2 follow opposite trends compared to AMA-5 and AMA-10, with the first pair incurring less

overhead when capacity is tight, but then increasingly more as capacity becomes abundant. This is due to two reasons. On the one hand, the number of migrations, and that of (successful) capacity reservations in ILA, increases more steeply for low network awareness, as shown in Fig. 6. On the other hand, the number of unsuccessful reservations, initially larger for the greater awareness, generally decreases with increasing capacity. This is confirmed in Fig. 8 which shows the percentage of control messages that resulted in a back-off. The net effect results in the observed behavior.

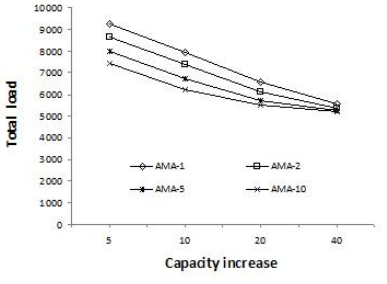

Fig. 5. Total load vs. capacity increase (50 nodes, application mix)

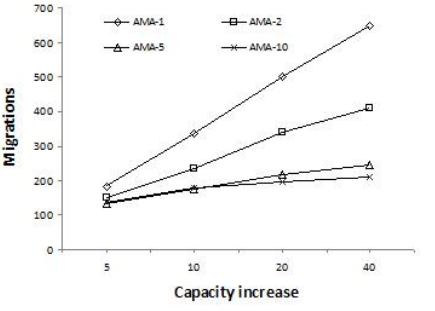

Fig. 6. Migrations vs. capacity increase (50 nodes, application mix)

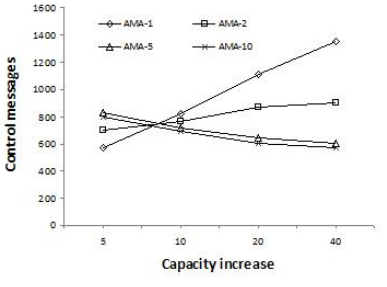

Fig. 7. Control overhead vs. capacity increase (50 nodes, application mix)

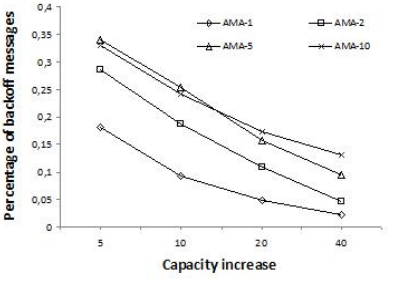

Fig. 8. Back-offs vs. capacity increase (50 nodes, application mix)

Fig. 7 plots the control messages generated by the algorithms. It is interesting to node that AMA-1 and AMA-2 follow opposite trends compared to AMA-5 and AMA-10, with the first pair incurring less overhead than the second pair when capacity is tight but then increasingly more as capacity becomes more abundant. This is due to two reasons. On the one hand, the number of migrations increases more steeply for lower network awareness, as shown in Fig. 6. On the other hand, the probability of unsuccessful capacity reservations (recall that ILA is used) generally becomes smaller as capacity increases. This is confirmed in Fig. 8 which plots the ratio of control messages that correspond to failed locking attempts to the total number of generated control messages. The net effect is in favor of greater network awareness, hence the observed behavior.

6.5 Result Summary

Based on the presented results we can state that: (i) AMA-*k* achieves close to optimal performance when there are no capacity constraints; (ii) with capacity constraints, AMA-*k* considerably improves agent placement from an initial random placement; (iii) greater network awareness leads to better placements while requiring fewer migrations, but this performance advantage shrinks rather quickly for larger values of *k*; (iv) the ILA scheme scales better than ILB, and in fact leads to better placements for increased network awareness when node capacity is scarce.

7 Related Work

Various systems developed for mobile code applications support agent-migration. For instance, Agilla [1] and Pushpin [6] consider 1-hop neighbors as possible candidates for hosting mobile code, while SmartMessages [2], DFuse [7] and MagnetOS [8] also consider up to k-hop neighbors. We also propose algorithms for 1-hop and *k*-hop awareness but with more emphasis in dealing with node capacity constraints using two different reservation schemes which are studied for different scenarios.

Placement problems have been investigated under various contexts, e.g., file allocation [9], service placement [10] and task allocation [11] to name a few. The work closest to ours (in algorithmic nature) is perhaps [12] where a distributed algorithm is presented for migrating data objects towards the center of gravity of client load. Our work differs mainly in that agents communicate with each other rather than with server-like entities. We also explore *k*-hop network awareness.

More specifically for WSNs, [13] discusses task allocation with the aim to minimize energy consumption, using a different application model and for the case where subtasks are created in the 1-hop neighborhood of their parent. In [14] the authors study the problem of data dissemination along a tree network, but the problem is different from the one discussed here. Agent migration is considered in [15] but with the aim of defining data collection paths rather than a (stable) placement. In previous work [16] we studied agent placement with focus on the admission of new agents while maximizing application lifetime, proposing centralized solutions.

8 Conclusions

In this paper we formulated the problem of placing cooperating mobile agents on nodes as to minimize the network load due to agent-level message traffic under node capacity constraints. We proposed and evaluated corresponding distributed algorithms for agent migration that can take advantage of basic routing-level information. Given their simplicity, these algorithms are suitable for resource constrained embedded systems. AMA-*k* combined with the ILA capacity inquiry and reservation scheme is a particularly attractive candidate since it achieves good results for relatively small (compared to the network diameter) values of *k*, incurring a modest communication overhead and being quite efficient in terms of memory and runtime complexity.

Acknowledgements

This work is funded by the 7th Framework Program of the European Community, project POBICOS, FP7-ICT-223984. Nikos Tziritas is supported in part by the Alexander S. Onassis Public Benefit Foundation in Greece.

References

1. Fok, L., Roman, G., Lu, C.: Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications. In: Proc. ICDCS 2005 (2005)
2. Kang, P., Borcea, C., Xu, G., Saxena, A., Kremer, U., Iftode, L.: Smart Messages: A Distributed Computing Platform for Networks of Embedded Systems. The Computer Journal 47(4) (2004)
3. Domaszewicz, J., Roj, M., Pruszkowski, A., Golanski, M., Kacperski, K.: ROVERS: Pervasive Computing Platform for Heterogeneous Sensor-Actuator Networks. In: Proc. WoWMoM 2006 (2006)
4. POBICOS project web site, `http://www.ict-pobicos.eu`
5. Loukopoulos, T., Tziritas, N., Lampsas, P., Lalis, S.: Implementing Replica Place-ments: Feasibility and Cost Minimization. In: Proc. IPDPS 2007 (2007)
6. Lifton, J., Seetharam, D., Broxton, M., Paradiso, J.A.: Pushpin Computing System Overview: A Platform for Distributed, Embedded, Ubiquitous Sensor Networks. In: Mattern, F., Naghshineh, M. (eds.) PERVASIVE 2002. LNCS, vol. 2414, p. 139. Springer, Heidelberg (2002)
7. Ramachandran, U., Kumar, R., Wolenetz, M., Cooper, B., Agarwalla, B., Shin, J., Hutto, P., Paul, A.: Dynamic Data Fusion for Future Sensor Networks. ACM Trans. Sen. Netw. 2(3) (2006)
8. Liu, H., Roeder, T., Walsh, K., Barr, R., Sirer, E.G.: Design and Implementation of a Single System Image Operating System for Ad Hoc Networks. In: Proc. MobiSys 2005 (2005)
9. Khan, S., Ahmad, I.: A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids. IEEE TPDS 20(3) (2009)
10. Laoutaris, N., Smaragdakis, G., Oikonomou, K., Stavrakakis, I., Bestavros, A.: Distributed Placement of Service Facilities in Large-scale Networks. In: Proc. INFOCOM 2007 (2007)
11. Agarwal, T., Sharma, A., Laxmikant, A., Kale, L.: Topology-aware Task Mapping for Reducing Communication Contention on Large Parallel Machines. In: Proc. IPDPS 2006 (2006)
12. Wolfson, O., Jajodia, S., Huang, Y.: An Adaptive Data Replication Algorithm. ACM Trans. on Database Systems 22(4) (1997)
13. Tian, Y., Boangoat, J., Ekici, E., Özgüner, F.: Real-time Task Mapping and Schedul-ing for Collaborative in-Network Processing in DVS-enabled Wireless Sensor Networks. In: Proc. IPDPS 2006 (2006)
14. Kim, H., Abdelzaher, T., Kwon, W.: Dynamic Delay-constrained Minimum-energy Dissemination in Wireless Sensor Networks. ACM Trans. on Embedded Computing Systems 4(3) (2005)
15. Wu, Q., Rao, N., Barhen, J., Iyenger, S., Vaishnavi, V., Qi, H., Chakrabarty, K.: On Computing Mobile Agent Routes for Data Fusion in Distributed Sensor Networks. IEEE Trans. on Knowledge and Data Engineering 16(6) (2004)
16. Tziritas, N., Loukopoulos, T., Lalis, S., Lampsas, P.: Agent Placement in Wireless Embedded Systems: Memory Space and Energy Optimizations. In: Proc. Int. Workshop on Performance Modeling, Evaluation and Optimization of Ubiquitous Computing and Networked Systems (2010)