A Self-adaptable Agent System for Efficient Information Gathering

Antonio Liotta[1], George Pavlou[1], Graham Knight[2]

[1] CCSR, University of Surrey, Guildford, GU2 7XH, UK
{A.Liotta, G.Pavlou}@eim.surrey.ac.uk
[2] Computer Science Department, University College London, London WC1 6BT, UK
G.Knight@cs.ucl.ac.uk

Abstract. As networks become all-pervasive the importance of efficient information gathering for purposes such as monitoring, fault diagnosis, and performance evaluation can only increase. Extracting information out of large-scale, dynamic networked systems is becoming increasingly difficult. Distributed monitoring systems based on static object technologies such as CORBA and Java-RMI can cope with scalability problems only to a limited extent. They are not well suited to monitoring systems that are both very large and highly dynamic because the monitoring logic, although distributed, is statically pre-determined at design time. The paper presents an *active distributed monitoring* system based on mobile agents. Agents act as area managers which are not bound to any particular network node and can sense the network, estimate better locations, and migrate in order to pursue location optimality. Simulations demonstrate the capability of this approach to cope with large-scale systems and changing network conditions. The limitations of our approach are also discussed in comparison to more conventional monitoring systems. **Keywords**. Self-adaptable monitoring; Scalable Information Gathering; Adaptable Information Gathering; Mobile Agents.

1 Introduction

While the size of networked systems grows at an incredible pace, it becomes increasingly difficult to extract information out of those systems. Networked systems and even the networks themselves need constant monitoring and probing for the purposes of management, particularly for fault diagnosis and performance evaluation. For instance, network monitoring entails the collection of traffic information used for a variety of performance management activities such as capacity planning and traffic flow predictions, bottleneck and congestion identification, quality of service monitoring for services based on service level agreements, etc. In this case, a key aspect is that collection of traffic information should be supported in a timely manner, so that reaction to performance problems is possible, and without incurring excessive additional traffic on the managed network. In this article, we highlight the limitations of existing solutions and propose an approach that uses the emerging paradigm of mobile software agents.

Conventionally, information is gathered following a *centralized* paradigm, where most of the intelligence is concentrated in a single management station which is in

charge of collecting and processing information. This approach has been widely criticized for its limited responsiveness, accuracy and scalability. Typically, the system is partitioned into smaller areas, each of which is monitored by a separate 'area' manager. More generally, *static decentralized monitoring* is realized with an *n*-level hierarchy of area managers. This is a static approach because the locations of the area managers are computed off-line and do not change after deployment. It scales better than its centralized counterpart but still lacks the adaptability necessary to cope with the frequently changing conditions of large-scale, dynamic networked systems.

To support such adaptation, the information gathering system needs to be able to sense the network state, which is generally dynamic and transient, and react appropriately. Numerous efforts have been devoted to monitoring and probing networks according to a static decentralized approach. Very few, however, have pursued a more dynamic approach to distributed monitoring where the area managers can actually re-locate themselves at run time to adapt to changing conditions in the underlying monitored system. We term this approach *active distributed monitoring* and present our views on how it can be realized with Mobile Agents (MAs).

Analogous requirements to those assumed herein have been addressed using an approach based on static co-operating agents to build a scalable network measurement infrastructure [1]. The possible advantages of using agent mobility for network management have been discussed extensively in the agent community and are detailed in [2]. Some of these advantages are reduction in network traffic, increased responsiveness, and support for disconnected computing. Furthermore, the authors elaborate on possible applications of MAs to fault management, remote diagnostics, configuration management and performance management.

However, most commonly, in the context of management, MAs are not exploited to the full extent of their capabilities. In fact, the majority of examples presented in the literature use MAs more simply as a mechanism to realize dynamic programmability of remote elements according to the Management by Delegation (MbD) concept, discussed in [3]. We have carried out some work in that direction, identifying key performance issues and studying a possible implementation of MbD based on MAs which we have termed *constrained agent mobility* [4]. We have then shown the benefits (in terms of added flexibility and dynamic re-programmability) of constrained mobility in the particular context of network performance monitoring by comparing it with conventional approaches based on static distributed object technologies [5] such as CORBA and Java-RMI.

MbD and constrained mobility may be realized with agents bound to *single-hop* mobility, from manager to remote elements. What is not commonly exploited in management is the agent *multiple-hop* capability. In the work presented herein we elaborate on the benefits of *agent weak mobility* [6] –the ability of an agent of carrying *code* and *data* when traveling from node to node– for adaptable distributed monitoring. Conversely, *agent strong mobility* is the ability of carrying also execution state, a property which is not believed to be suited to management applications.

Given our proposal to use MAs as area monitoring stations, a distributed algorithm is required to compute the agent locations both initially and at run time. During the execution of the monitoring task agents will need to sense their environment and take actions in order to adapt to changing conditions and, by doing so, maintain location optimality. Optimality in this case concerns the minimization of the network traffic

incurred by the agent-based monitoring system and of the latency in collecting the necessary information.

A similar problem regarding the optimal placement of *p* servers in a large network has been studied since the early seventies. This belongs to the class of *p*-center and *p*-median problems, both NP-complete when striving for optimality [7, 8]. Approximate polynomial algorithms, such as the *lagrangian* algorithm [8], have been proposed but none of them suites the requirement of our agent system. Proposed algorithms are centralized, requiring the network distance matrix at the main monitoring distance. While this is less of a problem in off-line calculations for medium to long-term optimal locations, it becomes an important problem for active distributed solutions in which optimal locations need to be (re)-calculated by the agents themselves. In this case the monitoring station should retain an up-to-date version of whole network topology, which obviously is an unrealistic requirement for large-scale, dynamic networked systems.

In this article we describe our solution to the agent location problem, evaluate its computational characteristics, and demonstrate by computer simulation some of the important features of the proposed agent-based distributed monitoring system. Our algorithm relies on agents learning about the network topology through node routing-table information which is accessed through standard management interfaces. The monitoring system is initially deployed through a "clone and send" process starting at the centralized network-wide station. The same algorithm is also used by the agents to adapt to network changes through migration. Key features of this algorithm are its distributed nature, i.e. each agent carries and runs the algorithm, and its low computational complexity and typical computational time. We discuss the scalability of our approach and its ability to adapt to network congestion and faults.

2 Real-time computation of the agent locations

The agent location problem consists of two phases. Initially, we need to determine what is the appropriate number of agents for a given monitoring problem and compute the location of each of those agents. Subsequently, upon agent deployment the agent system needs to be able to self-regulate in order to adapt to changing conditions. This is achieved by triggering agent migration in a controlled fashion to avoid instability due to continuous agent migration.

In the proposed solution, the location of area managers is neither fixed nor pre-determined at design time. Area managers are realized with mobile agents, simple autonomous software entities that, having access to network routing information, can adapt and roam through the network. The distributed monitoring system is deployed by progressively partitioning the network and by populating each partition with monitoring agents. We assume the existence of an agent system supporting *weak mobility* and *agent cloning* –i.e. the ability of agents to create and dispatch copies, or 'clones', of themselves. Agents are assumed to have access to routing information obtainable from network routers through standard network management interfaces.

2.1 Agent Deployment

The agent deployment process is illustrated Fig. 1.

The *number* and *location* of mobile agents is computed by subsequently comparing the monitoring task parameters with routing information extracted from network routers. Starting from the monitoring station, the list of monitored objects (MOs) is matched against *next-hop* addresses and *routing costs* to reach those MOs from the current location. This simple matching operation is sufficient for the agent to create a first partitioning of the network. A number of agents equal to the number of partitions is cloned. Each agent is assigned to a different partition and is configured to monitor the subset of the MOs belonging to that partition. Then, each agent autonomously resumes the "partitioning & cloning" process that ends when the number of MOs per partition falls below a given heuristic threshold.

```
1 Get monitoring task specs (at the monitoring station)
2 Generate 1 MA implementing the task (set MA parameters to the values
extracted from task spec)
3 Extract list of MOs from current MA
4 For each MO extract routing info from local router
  * get next-hop node id from current MA location to MO
  * get cost to reach MO from current MA location
5 Estimate cost for current MA to monitor its MOs
6 Use cost to compute number of MAs to be cloned by MA and clone them
7 Decompose task of current MA into a suitable number of sub-tasks
8 For each current MA
  * set its task to one of the above sub-tasks
  * set its list of MOs to a disjoint subset of the total current MOs
  * estimate cost to start monitoring from current location
  * estimate cost to monitor from one neighbor location
  * IF (lowest cost is from current location)
            THEN start MA
  * ELSE {
            migrate to the cheapest location
            GOTO 3
          }
```

Fig. 1. Proposed agent location algorithm.

The algorithm can be further illustrated by showing the basic steps performed in the case of the simple network depicted in Fig. 2i. Those steps are depicted in Fig 3. Initially, the manager delegates a given task to one agent and starts it at the monitoring station (a). By extracting routing information from the local router and matching them with the list of MOs, this agent estimates the need for an extra agent. An agent is thus cloned, and the original task is decomposed into two sub-tasks, including the redistribution of the MOs between the two agents (b). Then each agent autonomously searches its best location and migrates to it (c). The agent in location 1 is now ready to start since it has estimated that its current location is the one with minimum cost. In contrast, the agent in location 2 decides to share its task with another agent and clones it (d). The decomposition/migration process starts again leading to one agent running in node 2 and the other migrating to node 8 (e). Eventually, the agent in location 8 has found its cheapest location and start executing (f).

It should be noted that the use of cloning results in minimal traffic around the monitoring station. In fact, only two agents leave the station, although the resulting number of agents is three. The cloning algorithm is executed in a distributed fashion (on nodes 1, 2, and 8). Finally, the processing is performed in parallel among nodes at the same level (1 and 2). This algorithm is computed dynamically in the sense that the final agent location depends critically on the network status detected at deployment time.

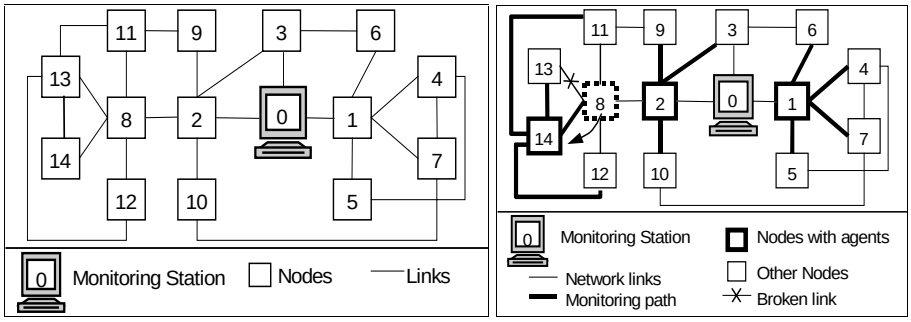

Fig. 2. i) Sample network topology. ii) Example adaptation through agent migration following a link failure.

2.2 Run time Agent Self-regulation

The ability of a monitoring system to adapt to network changes is a very attractive property, especially in view of the dynamic behavior of current and future networks. Network congestion and failures, along with mobile computing result in rapidly changing network logical topologies.

The conventional approach is to achieve adaptability by dynamically changing the routing tree rooted at the monitoring station. This is performed by the routing protocols. Consequently, as a result of congestion or failures, monitoring packets get re-routed through generally longer paths and both traffic and response times tend to deteriorate.

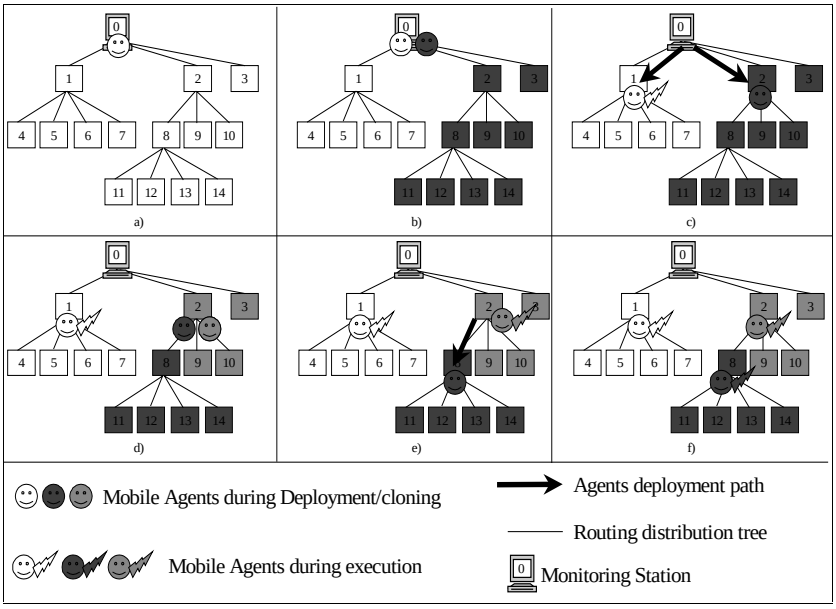

Fig. 3. Example agent deployment process.

In active monitoring, agents keep sensing the network during their operation and can periodically estimate the cost of alternative locations. Agent migration is triggered when the cost reduction justifies the migration overheads. In our implementation, agents adopt the same logic used during deployment time to sense the network and estimate costs associated to candidate neighbor nodes.

A simple example illustrating agent self-regulation in response to a link failure is depicted in Fig. 2ii. In this case, following a loss of connectivity between node 8 and 13, a new (longer) monitoring path is established between node 8 and node 13. As a result, the central node for the system partition comprising nodes {8, 11, 12, 13, and 14} becomes node 14. Hence, the agent originally located in node 8 will relocate to node 14, bringing the system back to optimality.

This adaptation strategy is based solely on local decisions. An agent knows which nodes belong to its partition and builds cost functions based on the information concerning those nodes, available at the local router. One can argue that the self-reconfiguration mechanism considered as a whole might suffer as a result of this myopic approach. On the other hand, agent myopia has the advantage of simplicity and reduced processing overheads. What the system cannot do is to apply global optimization strategies at run time.

Consequently, the agent system may gradually shift away from optimality if the monitoring task is relatively long and for extreme modifications of the network state. To provide adaptation to those situations we followed the simple approach of re-initiating the whole deployment process. This is more expensive than just migrating a subset of the agents because involves terminating all the agents and starting all over again. Agent re-deployment may be triggered periodically with a period which depends on the system dynamics. Alternatively, it could be triggered automatically by alarms or directly by a human operator.

In practice, our simulations with realistic network topologies (see sections below) has shown that agent re-deployment is not typically necessary because agents tends to end up precisely in the same locations in most cases.

Therefore, trade-off design choices between agent migration and periodic re-deployment are necessary for an efficient self-regulating system. In addition, other simple control mechanisms will contribute to the stability of the system. For instance, agents need to incorporate some inertial mechanism to prevent a situation in which minor, high-frequency fluctuations in the network trigger inconsiderate agent migration. Finally, more sophisticated control mechanisms may be considered such as run-time cloning or new agents to respond to rapid increase in system scale. These are not been included in the current prototype because we first tried to approach the location problem in a simple way. Run-time cloning will require mechanisms such as orphan control, containment of agent proliferations etc, which are out of the scope of this paper.

3 Evaluation Methodology

The proposed agent-based monitoring system has been evaluated from different points of view. First, the feasibility of the system depends critically on the agent deployment (or re-deployment) time. We assessed this aspect mathematically to be able to draw

conclusions not only on the asymptotic computational complexity of the deployment algorithm but also on typical deployment times under realistic conditions.

Having proved the feasibility of our algorithm we assesses its goodness through simulations. A crucial point was to run the algorithm for a set of realistic network topologies, composed of routers, links, and hosts. These have been generated using the GT-ITM topology generator [9, 10, 11]. In particular, transit-stub topologies resembling the Internet topology and having 16, 25, 32, 50, 64, 75, and 100 nodes respectively, have been generated in order to assess the sensitivity of the location algorithm to network size. For each network size, simulations have been repeated at least 10 times over randomly generated networks characterized by identical topological features. This was done to guarantee statistical significance of the results. Example 50-node topologies are reported in Fig. 4. You can notice that the actual topologies are significantly different despite other topological features such as average node degree and network diameter are comparable.

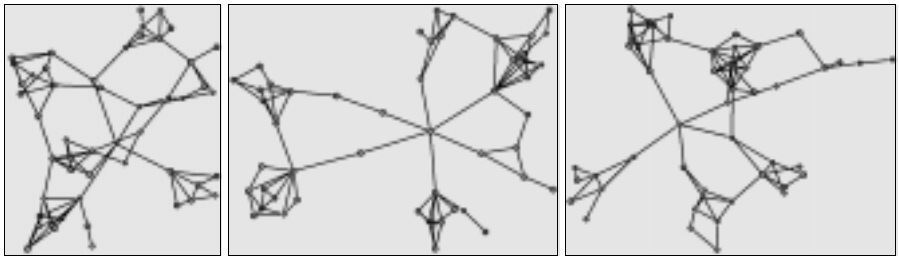

Fig. 4. Example 50-node randomly generated network topologies.

In order to simulate IP network and protocol behavior we have adopted the NS-2 simulator from U.C. Berkeley/LBNL [12] and extended it with Mobile Agent capabilities. Agent migration and cloning have been implemented along with the actual agent location algorithm, which is incorporated in each agent. This algorithm has been optimized to minimize the total incurred monitoring traffic. Total hop-distance and maximum weighted distances have been measured for increasing “agents to number of monitored objects” ratios. Those metrics are directly related to the total traffic incurred by the monitoring system and to its response time.

An important parameter we measured was the distance from optimality. To assess how far from optimality our agents ended up we computed the agent locations using three different algorithms: 1) the proposed algorithm; 2) the lagrangian algorithm [8]; and 3) a random location. The lagrangian algorithm is provably near-optimal; hence, by achieving smaller traffic and response time than the ones obtained with it we proved near-optimality of our algorithm. The lagrangian algorithm was computed using the software package SITATION [8]. We also generated the agent location randomly to emulate the worse possible agent distribution.

An important feature of distributed monitoring systems is their ability to scale better than their centralized counterparts. To quantify the potential benefits we have measured traffic and response time for increasing values of polling rate, number of monitored nodes, network diameter, and number of agents. Due to lack of space though we report only the first case.

Finally, we started studying the self-reconfigurability of our agent system by simulating various conditions in which link failures led to increased traffic and response time. We deployed the agent system before the failures; then generated link failures at random locations; assessed the costs associated to agent migration; and finally measured traffic and response time after re-configuration. We repeated the same experiment several times for statistical significance.

4 Adaptation through re-deployment

4.1 Agent Deployment Timescale

The agent location is actually computed during agent deployment. Hence, the algorithmic asymptotic complexity can be estimated by looking at the predominant factors involved from start up until all agents are deployed.

Steps 1-2 of Fig. 1 are performed at the monitoring station, at start up time. Their predominant factor is the cloning time, $CLON_{time}$. In contrast, steps 3-8 may be repeated at subsequent levels of the routing distribution tree (rooted at the monitoring station). They will be repeated at most $R(u)$ times, whereby $R(u)$ is the network radius. Agents running at the same level of the distribution tree, execute independently from each other, in separate physical locations. Hence the computational complexity of the location algorithm can be determined by considering the part that is inherently sequential. Therefore, the complexity is $R(u)$ times the complexity of steps 3-8.

Upon arriving at a node, an agent needs to be de-serialized and instantiated, before executing from step 3. This operation takes a constant time, $DESERIL_{time}$. Steps 3-4 require a number of iterations equal, at most, to the total number of monitored nodes. The dominant cost for each iteration is given by the look-up operation to the routing table to extract the *next_hop* and the *cost* values. Thus, the total contribution of steps 3-4 is $c*O(N)$, where c accounts for one look-up time. Step 5 involves a number of iterations which, in the worst case, is equal to the maximum node degree, δ_{max} that in typical networks is significantly smaller than the number of nodes and, typically, does not increase with N. The iterations of steps 6-8 are actually performed as part of steps 5 and in the worst case involve the process of cloning and configuring δ_{max} new agents. Cloning will take a constant time, $CLON_{time}$; the reassignment of the monitored nodes takes a constant time too because it reuses information initially processed during Steps 3-4. Finally, each new agent will require a serialization time, $SERIAL_{time}$ before being sent to its destination. The latter will add a forwarding delay, $FORW_{time}$ and a transmission time, $TRANSM_{time}$.

Therefore the agent deployment time, $DEPL_{time}$ that actually coincides with the time to compute the agent location algorithm, can be expressed as: $DEPL_{time} = \{DESERIL_{time} + c*O(N) + \delta_{max} * [CLON_{time} + SERIAL_{time}] + TRANSM_{time} + FORW_{time}\}*O(R(u)) = c_1 *O(N * R(u)) + c_2 * O(R(u)) \propto O(N*R(u))$.

In practice, c_1 is of the order of at most 10E-6 seconds, since the current technology allows for a number of look-up operations of at least 10E6 per second. c_2 is in the order of seconds since with current mobile agent platforms $[TRANSM_{time} + FORW_{time}]$ is typically in the order of 10E-3 to 10E-1 seconds and $[DESERIL_{time} + CLON_{time} + SERIAL_{time}]$ is in the order of seconds or fraction of seconds [5]. Therefore, if $N << 10E6$ then $[c*O(N)] << \{DESERIL_{time} + \delta_{max} *[CLON_{time} + SERIAL_{time}] + TRANSM_{time} +$

$FORW_{time}$} and, consequently, $DEPL_{time} \approx c_2 * O(R(u))$. In this case the deployment term will predominate over the computational one and $DEPL_{time}$ will be in the order of seconds times $O(R(u))$.

4.2 Distance from Optimality

The distance from optimality of the proposed location algorithm can be evaluated by observing the plots of Fig. 5. The total hop-distance is directly related to the total steady-state monitoring traffic. It can be observed that the proposed location algorithm leads to traffic values that are always smaller than those that would be achieved with the lagrangian algorithm, which is provably near-optimal. Hence, our agent-based algorithm is near-optimal too. In particular, a percentage improvement in the range of 0-3% was measured. It should be stressed once again that the lagrangian algorithm cannot be used to solve the agent location algorithm for the reasons already mentioned in the introduction.

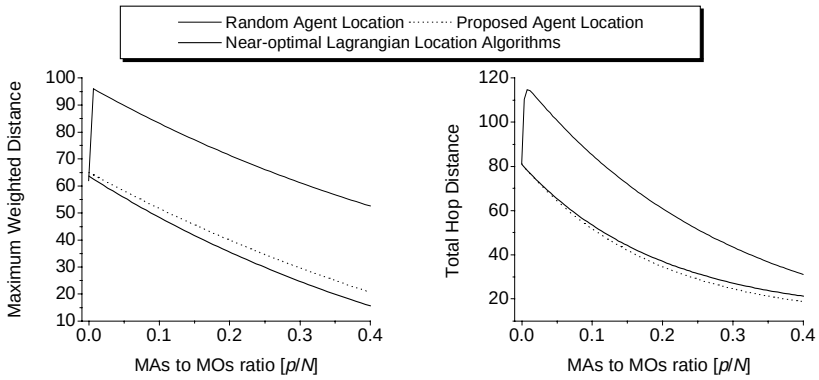

Fig. 5. Distance from near-optimality.

It should be noted that, for the sake of completeness, we simulated situations characterized by up to a large number of agents (p/N=0.4). However for a more efficient resource utilization, typical "agents to nodes" ratios are envisioned to be much smaller ($p/N \approx 0.1$). The fact that the total hop-distance achieved by placing the agents in a random fashion is very far from our near-optimal solution (38-48% difference for $p/N<0.1$) provides another good justification for the adoption of the agent-based approach. The percentage reduction in traffic with respect to centralized polling (p/N=0) is also significant. For instance, for p/N=0.1 the reduction in traffic will be greater than 30% and will increase monotonically with p/N.

Finally, the fact that the three curves tend to converge for large values of p/N is not unexpected since when p/N=1 the number of agents equals the number of nodes. Hence, each of the three location algorithm will equally succeed in placing the agent evenly. The plot which reports the maximum weighted distance (directly related to

response time) for the three location algorithms is qualitatively analogous to the previous one. However, in this case the agent location curve, though very close to the near-optimal one, does not exhibit any inferior value. In particular, the distance from near-optimality is 0-5% for $p/N<0.1$. This result was expected since the simulated agent location algorithm was optimized to minimize traffic, not response time. Further simulations, not reported here for brevity, proved that near-optimality with respect to response time can be achieved with appropriate alterations to the agent algorithm.

4.3 Scalability

This section evaluates the scalability of the proposed monitoring system from a different viewpoint than the one of Section 4.2. We previously assessed how well the agent deployment algorithm scaled to draw conclusions on its viability. Herein, we evaluate scalability at steady-state by comparing the agent monitoring system with a conventional centralized system. Expectedly, an improvement in performance is achieved with the former approach due to its intrinsic distributed nature. However, the results of our simulations provide a quantitative evaluation.

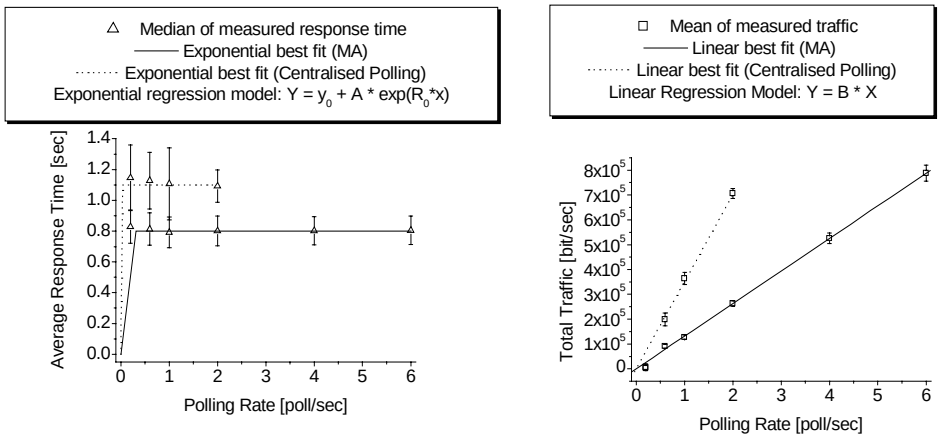

Fig. 6. Scalability.

Fig. 6 shows traffic and response time measurements achieved with centralized and distributed polling-based monitoring, respectively. Increasing values of polling rate are required for larger accuracy and timeliness, but incur increasing volume of traffic. In our scenario $p/N=0.1$, whilst network diameter and average node degree are kept constant, hence the linear behavior. It can be observed that the agent solution leads to an approximate 50% reduction in traffic and 28% reduction in response time. Another aspect of scalability is the maximum polling rate that can be sustained by the network. Our simulations showed that the agent solution could sustain polling rates of the order of 200% larger than its centralized counterpart. We then assessed the sensitivity to *p/N*,

keeping all the other parameter unchanged. This time the curves, not shown for brevity, exhibited a non linear behavior. Both traffic and response time decreased significantly for agent configuration having $0<p/N<0.15$. However, for larger values of p/N the improvement was negligible. We concluded that a larger portion of agents is neither convenient nor useful. In fact, the larger is the number of agents, the larger the agent deployment overheads.

5 Adaptation through migration

In this section we present some of our simulation results aimed at evaluating the adaptability of the agent system in face of changing conditions. Agent migration overheads are a major limiting factor but are typically followed by significant improvement in terms of reduced monitoring traffic and responsiveness.

5.1 Migration Overheads

Agent migration overheads are predominated by agent migration time and traffic. In typical general-purpose MA platforms migration time varies in the range between hundreds of milliseconds to seconds [5, 13]. This means that the agent system needs to manage a transient time associated to agent migration in the order of seconds. To improve the persistency of the monitoring system a possible solution would be to implement agent migration through cloning. Instead of migrating, an agent clones another agent and dispatches it to its intended destination. Upon arrival to the target node, the child agent will terminate its parent. This is not a feasible solution for every kind of monitoring task but could often lead to significant improvements.

Another migration overhead is associated with migration traffic. This depends on the agent size which is in turn a function of the complexity of agent logic and of the amount of data transported with the agent. Our agents do not support strong mobility; hence, they do not have to carry the burden of the execution state. In addition, they are designed following the principle of simplicity; then they are relatively small in size.

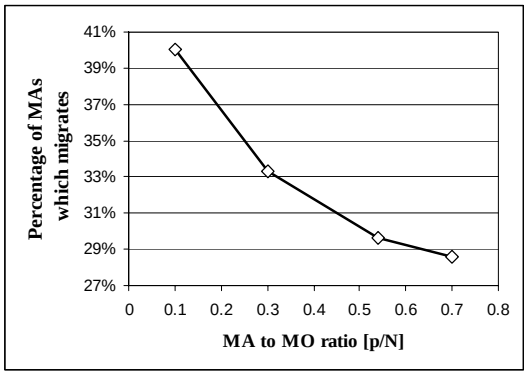

Fig. 7. Impact of total number of agents on percentage of agent migration occurrences.

An important design choice is the number of agents initially deployed. In fact the more agents we deploy, the higher the deployment (and re-deployment) overheads will be. A large number of agents also means a larger consumption of computational and memory resources in the hosting nodes. The benefit of a large number of agents is related to a higher level of distribution, followed by generally better steady-state performance of the monitoring system. Another advantage is that, as the number of agent increases, the percentage of agents that need to migrate in face of changing network conditions tends to decrease more than linearly, as demonstrated by our simulation results reported in Fig. 7.

5.2 Migration Benefits

We have simulated a simple scenario in which 2 links located in the vicinity of the central monitoring station fail. Traffic and response time were measured before the failure. After the failure, the routing protocol readjusted the routing tables and full connectivity was achieved. In addition, the agent system reconfigured itself by relocating some of the agents. Steady-state traffic and response time were measured again. Simulations were subsequently repeated for 10 different randomly generated topologies characterized by comparable topological features. Each time a couple of faults was generated randomly and results were averaged.

Fig. 8 shows the snapshot of those two performance indicators (traffic and response time) taken before and after the link failure, respectively. With the centralized polling solution (p = 0), both request and response packets get re-routed through longer paths. Consequently, both traffic and response time increase significantly –they almost doubled in our scenario. On the contrary, with the agent system the performance degradation at steady state is in the order of 5-10%.

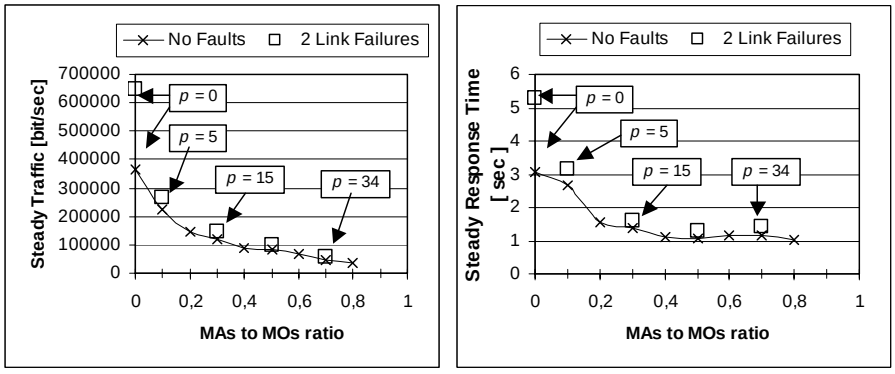

Fig. 8. Self-adaptation through agent migration.

It should be noticed that, though more extensive simulation will be needed before more generalized conclusions can be drawn, the results achieved so far are very promising. We shall investigate what happens when the number of faults increases to assess the robustness of our system. We have not simulated scenarios in which a fault leads to a

temporary loss of connectivity. Moreover, it will be interesting to conduct more thorough simulations to assess the stability of the agent system.

6 Concluding remarks

In this paper we have presented our progress towards the design of a self-regulating, distributed monitoring system based on mobile agents. While a lot of work has addressed the problem of building scalable, distributed monitoring systems based on the Management by Delegation principles, not much has been done to pursue adaptability in the context of large-scale, dynamic networked systems. We believe that adaptable information gathering is a crucial feature, in view of the pervasiveness of network-centric applications. The interest created by architectures such as SUN's Jini [14] shows that the scenario in which a relatively large number of simple devices will be accessible across the net is becoming realistic. This introduces a new dimension to networked systems which will become more and more dynamic as we also observe a shift towards all-IP, integrated fixed and mobile network infrastructures.

Of additional relevance to this article is the fact that Jini devices can host mobile code, a feature which would have been unthinkable just a few years ago. However, code mobility represents a serious paradigm shift in the management arena which has not yet found widespread acceptance in the community. It is often claimed that this is due to persistent security and safety concerns which are particularly critical in network and system management.

On the other hand, the benefits of code mobility tend to be undermined by the scarcity of established design methodologies which suit management applications. Code mobility adds a degree of freedom which is hardly conceivable if compared to the well standardized architectures and methodologies refined over the years. The work described herein aims at exploiting this extra degree of freedom. Our initial results are very promising in terms of improved scalability and flexibility achievable with the MA capabilities. We have discussed how agent *weak migration*, *autonomy*, *reactiveness*, and *cloning* can be employed to design a self-regulating monitoring system targeted to large-scale, highly dynamic networked systems. Another interesting property that might be worth investigating is agent *pro-activeness* to anticipate problems rather than just reacting to them.

Another comment concerns the comparison of the proposed algorithm with approaches based on static distributed object technologies such as CORBA and Java-RMI. If it was possible to accurately estimate the location of the area managers at system design time it would be significantly more efficient to realize area managers with static object technologies rather than MAs. Migration and cloning overheads would be avoided in such case. However, the static approach would not cater for the adaptability offered by the agent solution.

The relatively high costs associated to agent migration supported by general-purpose MA platforms give also an indication of the timescales over which adaptation might be effective. When agent migration times are in the order of a second, the agent system is able to compensate to changes within timescales larger than a second. On the other hand, steady-state performance and scalability will be comparable to those typical of systems based on static object technologies provided that effective methods are adopted to place those objects.

Acknowledgments

The work reported in this paper has formed part of the Software Dependent Systems Work Area of the Core II Research Programme of the Virtual Centre of Excellence in Mobile & Personal Communications, Mobile VCE, www.mobilevce.co.uk, whose funding support, including that of EPSRC, is gratefully acknowledged. More detailed technical reports on this research are available to Industrial Members of Mobile VCE.

References

1. Y. I. Wijata, D. Niehaus, V. S. Frost, "A Scalable Agent-based Network Measurement Infrastructure", *IEEE Communications Magazine*, (September 2000).
2. A. Bieszczad, B. Pagurek, T. White, "Mobile Agents for Network Management", *IEEE Communications Survey*, Vol.1, N.1, (Fourth Quarter 1998).
3. G. Goldszmidt, Y. Yemini, "Delegated Agents for Network Management". *IEEE Communications Magazine*, Vol.36 No.3, (March 1998).
4. C. Bohoris, A. Liotta, G. Pavlou, "Software Agent Constrained Mobility for Network Performance Monitoring", Proc. of the 6th IFIP *Conference on Intelligence in Networks* (SmartNet 2000), Vienna, Austria, ed. H.R. van As, pp. 367-387, Kluwer, (September 2000).
5. C. Bohoris. A. Liotta, G. Pavlou, "Evaluation of Constrained Mobility for Programmability in Network Management", Proc. of *DSOM 2000*, pp. 243-257 (December, 2000).
6. M. Baldi, G. P. Picco, "Evaluating the Tradeoffs of Mobile Code Paradigms in Network Management Applications", *ACM Transactions on Software Engineering and Methodology*, 20th International Conference on Software Engineering (ICSE '98), Kyoto, Japan, (April 1998).
7. B. C. Tansel, R. L. Francis, T. J. Lowe, "Location on Networks: a Survey", *Management Science*, Vol.29(4), pp.482-511, (April, 1983).
8. M. S. Daskin, "Network and Discrete Location", *Wiley*, (1995).
9. E.W. Zegura, K.L. Calvert, M.J. Donahoo, "A Quantitative Comparison of Graph-based Models for Internet Topology", *IEEE/ACM Transactions on Networking*, (1997).
10. E.W. Zegura, K.L. Calvert, S. Bhattacharjee, "How to Model an Internetwork", *IEEE INFOCOM 96*, San Francisco, CA, USA, (1996).
11. K.L. Calvert, M.B. Doar, E.W. Zegura, "Modeling Internet Topology", *IEEE Communications Magazine*, (June, 1997).
12. K. Fall, K. Varadhan, "NS Notes and Documents", UC Berkeley, (http:// www.isi.edu/ ~salehi/ ns_doc/) (October 1999).
13. G. Knight, R. Hazemi, "Mobile Agent based management in the INSERT project", *Journal of Network and System Management* (Mobile Agent-based Network and Service Management), Vol. 7 (3), (September, 1999).
14. J. Waldo, "The Jini Architecture for Network-centric Computing", *Communications of the ACM*, Vol. 42(7), pp. 76-82, (July, 1999).