Self-organizing Monitoring Agents for Hierarchical Event Correlation

Bin Zhang and Ehab Al-Shaer

School of Computer Science, Telecommunications and Information Systems
DePaul University, USA
{bzhang,ehab}@cs.depaul.edu

Abstract. Hierarchical event correlation is very important for distributed monitoring network and distributed system operations. In many large-scale distritbuted monitoring environments such as monitions senor networks for data aggregation, battlefield compact operations, and security events, an efficient hierarchical monitoring agent architecture must be constructed to facilitate event reporting and correlation utilizing the spacial relation between events and agents with minimum delay and cost in the network. However, due to the significant agent communication and management overhead in organzine agents in distributed monitoring, many of the existing approaching become inefficient or hard to deploy. In this paper, we propose a topology-aware hierarchical agent architecture construction technique that minimizes the monitoring cost while considering the underlying network topology and agent capabilities. The agent architecture construction is performed in a purely decentralized fashion based on the agents' local knowledge with minimal communication and no central node support.

1 Introduction

A Distributed network monitoring (DNM) system has become an indispensable component for current enterprise network. The high speed and large scale properties of enterprise network have imposed new requirements on DNM system. The DNM system should not only detect and transfer the network events efficiently, but also limit event propagation to save computing and network resources. A DNM system is composed of a set of distributed monitoring agents which cooperate together to fulfill the monitoring tasks. These agents form an architecture which determines the event delivery and aggregation path in the network. So DNM architecture is crucial to the system performance. Many DNM systems have been proposed in the past [15,3,6]. These works focus mainly on the network events detection and correlation models, dynamic monitoring job update and programmable action interface. However, the mechanism of constructing and maintaining the monitoring agent architecture was insufficiently addressed. In many DNM systems, the agent architectures are static and sometimes manually constructed and maintained. In other DNM systems, the agent architectures are constructed based on the logical relations between agents, which may not reflect the underlying network topology or agents' capabilities. These limitations

A. Clemm, L.Z. Granville, and R. Stadler (Eds.): DSOM 2007, LNCS 4785, pp. 13–24, 2007.

impose long construction time, low efficiency and unbalance monitoring task distribution in DNM systems.

In this paper, we introduce distributed self-organized topology-aware agents for constructing a hierarchical distributed monitoring architecture. We describe a number of algorithms and management protocols that enable agents to cooperate together to build a hierarchical architecture in a decentralized fashion. Agents work collaboratively but based on their local knowledge to select the most suitable set of leaders in each level in the hierarchy such that the monitoring information delivery cost is minimized. No central node or global knowledge is assumed in this work and the number of messages exchanged between agents is minimized by using scope-controlled multicast. Our proposed work takes into consideration of spacial event correlation which is important for many distributed applications such as security alarm correlation and sensor network data aggregation. In addition, topology-aware hierarchical correlation does not only restrict event propagation but it also allows sensors to save energy in transmitting events during the correlation/aggrtegation operation.

This paper is organized as follows. In section 2, we describe and formalize the problem that we address in this work. Section 3 describes our heuristic algorithm to develop hierarchical monitoring architecture using decentralized self-organizing agents. In section 4, we evaluate this approach and show the simulation results. Section 5 compares our work with related works. In section 6, we give the conclusion and identify the future work.

2 System Model and Problem Definition

A large number of monitoring agents might be distributed in a large-scale enterprise network. To address the scalability problem of having a large amount of agents in the distributed monitoring system, hierarchical monitoring structure is used. Agents are organized into different layers such that each layer contains a set of agents that are selected as leaders to which other agents report their events. Each leader filters and aggregates the incoming event and reports to higher level agent. In order to improve the system performance and reduce events propagation, the agent architecture should not only consider the underlying network topology and agent processing capability, but also the event traffic rate generated by each agent. Thus, in each layer, we want to find the optimal subset of agents as leaders and assign each non-leader agent to the most suitable leader. The assignment of an agent to a leader is associated with a communication cost, that is equal to the traffic rate of that agent multiplied by the distance (e.g., number of hops) from that agent to its leader. Our objective in each layer is to find the optimal subset of leaders and the assignment that can minimize the total communication cost while respecting the leaders' processing capability. Thus, the hierarchical agent architecture construction can be viewed as a set of recursive solutions to the Leader Selection and Assignment Problem (LSAP) described above.

Suppose we have a set of n monitoring agents in one layer. Let d_{ij} denote the distance between agent i and agent j. Let c_i denote the processing capability of

agent i, which is decided based on the CPU speed and memory. Let b_i represent the event rate of agent i, which is defined by $b_i = \theta\mu_i + \sigma_i$. Here, μ_i and σ_i denote the mean and standard deviation of event rate of node i respectively. The LSAP at each layer can be formalized as an integer linear program:

$$minimize \quad \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}b_i \tag{1}$$

subject to:

$$\sum_{i=1}^{n} b_i x_{ij} < c_j \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \tag{3}$$

$$x_{ij} \leq y_j \tag{4}$$

$$\sum_{j=1}^{n} y_i \leq \lceil \frac{\sum_{i=1}^{n} b_i}{\sum_{i=1}^{n} c_i/n} \rceil \tag{5}$$

$$x_{ij} \in \{0, 1\} \tag{6}$$

$$y_j \in \{0, 1\} \tag{7}$$

The binary variable x_{ij} represents whether agent i treats agent j as its leader. The binary variable y_j represents whether agent j is selected as a leader. The objective function (Eq.1) is to minimize the communication cost between agents and their leaders. The constraint in Eq.2 guarantees that each leader will not receive events more than a fraction of its capacity. The constraint Eq.3 restricts that each agent can report to one leader only. The relation between variable x_{ij} and y_j is defined by Eq. 4, which means if agent i report to agent j, then agent j must be a leader. The constraint Eq.5 is to specify that the total number of leaders in each layer can not be greater than the maximum number of agents required to accommodate all the event traffic coming from the this layer when using the average agent capacity ($\sum_{i=1}^{N} c_i/n$). We use this constraint to limit the depth of the monitoring hierarchy and thereby reduce the monitoring delay.

3 Optimal Self-organizing Monitoring Agents

Traditionally, to find a solution for this kind of optimization problem, we need one central agent that has the global knowledge of the network. When the network topology changes due to the nodes or network devices failure, the reconstruction has to be performed by the central node. This can easily causes a bottleneck in the monitoring system and creates a single point failure. In addition, to measure the distance between each agent pair and aggregate this information to the central node, a large number of probe messages ($O(N^2)$) should be exchanged. To overcome these shortcomings of using a centralized approach, we propose a decentralized agent architecture construction technique. The hierarchical structure is built in a bottom-up fashion such that all agents join the first

layer and then the optimal subset of leaders are collaboratively discovered and selected based on agents' local knowledge. The rest agents are assigned to these leaders. Then, the leaders from lower layer form the higher layer (ex. leaders of layer 1 form layer 2) and the leader selection and agent assignment process will be repeated at each layer till the agents at certain layer detect that there are no enough agents left and it is not necessary to further build the hierarchy, then these agents will directly report to the manager.

Based on the definition in section 2, we can see that the agents' architecture construction in each layer is an instance of the LSAP problem. So in this paper, We will focus the discussion on our distributed solution to LSAP. The LSAP can be proved to be NP-hard by a mapping from capacitated P-median problem (CPMP) [13]. Many approximation algorithm have been proposed for CPMP [14], but these approaches require a central node with global knowledge. So they are not suitable to large-scale monitoring architecture.

In the following section, we introduce our distributed solution for LASP and show how this algorithm can be repeatedly used to construct the hierarchical agent architecture. Our solution targets the monitoring architecture in large-scale enterprise network, where multicast and broadcast are normally available. Also, it is commonly known that in enterprise network, the hop counts relatively indicates the network delay between two nodes [5]. Our solution is totally distributed and efficient in terms of processing complexity and message exchange.

3.1 Distributed Leader Selection and Assignment Algorithm

Because the data from different monitoring nodes are often correlated and the correlation is usually stronger between data from nodes close to each other, we should select agents which have a small average distance to their members with high processing capacity as leaders. Thus, all data from nodes which share the same leader can be filtered, aggregated and correlated locally by the leader. Two tasks need to be accomplished in order to solve LSAP problem: (1) selecting the optimal subset of leaders, and (2) assigning each agent to the most suitable leader. Our algorithm combines these two tasks together into one step. The operation of our algorithm is divided into rounds and the length of round is represented by T. Before the agent architecture construction, each agent i computes its initial potential to become a leader (represented by P_i) based on its processing capability c_i. The P_i can be computed as $P_i = \eta \frac{c_i}{c_{max}}$. c_{max} is a constant which represents the maximal processing capability and $\eta < 1$ is used to control the initial value of P_i.

We assume agents can synchronously start the hierarchy construction about the same time based on a specific broadcast or control message sent by the system manager. At the beginning of each round, each agent sends out multicast search messages with limited scope as discussed below to inform other agents about its existence and status. The scope is used to control the distance the search message can travel in the network. We limit the scope to control the message propagation. The scope is implemented by setting the time-to-live (TTL) field in IP header of the search message equal to the scope value. The scope is increased by 1 at each

searching round. In the search message, each agent embeds its unique ID, event traffic rate b_i, and current scope. When one agent receives a search message from another agent, it computes the distance (i.e., number of hops) from the source by subtracting the TTL value in the IP header of received search message from the scope value in that search message. At the end of each round, each agent first updates its member list based on the search messages received so far. Because of the processing capacity limitation, each agent chooses its members based on the ascending order of distance, and at the same time the total traffic rate of its members should respect its capacity. Assume set M_i represents agent *i's* potential members, d_{ji} represents the distance between node i and j. The communication cost of agent i can be computed as follows: $cost_i = \sum_{j \in M_i} d_{ji} b_j$. Based on updated member list and communication cost, P_i value can be updated as follows:

$$(P_i)_{new} = (P_i)_{old}(1 + \alpha_1 \frac{s_i}{s_{max}} + \alpha_2(1 - \frac{cost_i/s_i}{\lambda})) \tag{8}$$

Here the parameter s_i represents the selected member list size of agent i, $s_i = |M_i|$. S_{max} is a constant which represents the maximal number of members any node can possibly have. And λ is a constant and used to normalize the average communication cost. In above equation, α_1 and α_2 are used to control how many percent each term contributes to P_i increase, $\alpha_1 + \alpha_2 = 1$. From equation 8, we can see that P_i value will increase inevitably at the end of each searching round. But how fast the P_i value increase is different for each agent. As member list size s_i increases (first term in Eq. 8), its contribution to P_i increase, and as the average communication cost $cost_i/s_i$ increases (second term in eq. 8), its contribution of it to P_i decreases. The more the members, the smaller the average communication cost, the faster the P_i value increases. Since the increase of P_i reflects both agent's processing capability and its distance to other agents, P_i is used as primary factor to decide whether an agent should become a leader. The agents with high P_i value has more possibility to become a leader.

3.2 Node Status

In this approach, each node has five status which can change based on its P_i value and its local knowledge.

- Unstructured: It means the agent doesn't belong to any leader's member list.
- Structured: It means the agent is included in some leader's member list.
- Leader candidate: The agent can be a leader candidate if the P_i value is larger than certain threshold P_T and less than 1.
- Leader: An agent becomes a leader if its P_i value reach 1.
- Isolated leader: An agent becomes an isolated leader when after R_T rounds it still can not become a leader or a member, or its $P_i = 1$ but it has no members.

In our algorithm, the initial status of all agents is unstructured. As the agent's P_i value increases at each search round, the agent status changes accordingly.

When an agent changes its status to a leader candidate or a leader, it sends out a multicast announcement to the selected members to make them stop searching. The leader candidate announcement is sent out periodically till this node becomes a leader. Leader node sends out beacon message (leader announcement) to its members periodically. The current members of a leader candidate or a leader are embedded in these announcements. If an unstructured agent receives a leader candidate announcement that includes its ID in the member list, it stops searching. If unstructured agent receives leader announcement that includes its ID, it change its status to structured and start membership determination.

The agent's status can also be changed reversely from a leader candidate to unstructured when it receives a leader candidate announcement from other agent which is more suitable to become a leader. Then how two leader candidates can compete with each other to determine which one is more suitable to become a leader? According to the definition of P_i, the node with higher P_i value is more suitable to become a leader. However, this is only true if they are closely related to make them competitors. The relation L_{ij} used to measure the competition relation between two candidate agents i and j can be computed as follow:

$$L_{ij} = \beta_1 \frac{|M_i \cap M_j|}{|M_i \cup M_j|} + \beta_2 (1 - \frac{d_{ij}}{d_{max}}) \tag{9}$$

M_i and M_j is the member list in agent i and j respectively, and d_{ij} is the distance (number of hops) between the agents i and j. Here d_{max} is a constant which stands for the largest distance in the network. β_1 and β_2 are used to control how many percent each term contributes to L_{ij} , $\beta_1 + \beta_2 = 1$. Intuitively, we can see that the more members these two leader candidates share, the smaller the network distance between them, the higher the competition relation value. Only when the relation L_{ij} between two agents i and j is larger than certain threshold L_T, these two agents become competitors. In this case, the agent with larger P_i prevails and keep its status as leader candidate. The other competitor changes its status to unstructured. The complete algorithm for leader selection can be found in [2]. This approach favors those agents which have small distances to their members and sufficient processing power to become leaders. To control the convergence speed, we set the threshold R_T as the maximal number of searching round for each node in the construction of one layer. After R_T round, if an agent still can not become a leader or a member, it will stop sending search message and declare itself as an isolated leader as will be discussed in section 3.3.

3.3 Agent Membership Determination

For unstructured node, member determination phase is the phase where the agent decides which leader to join in. When receives and included in a leader announcement, an unstructured agent waits for a short time T_d to see if other agents become leaders and include itself as member during that period. If it appears in multiple leaders' member list, the agent always chooses the closest leader by sending a multicast membership confirmation message with TTL equal to the distance to the farthest leader. This message causes other leaders or leader

candidates to remove this agent from their member lists. If a leader doesn't receive anything from its member, it assumes that agent agrees to be its member. For leader candidate, the member determination phase is the phase where the leader candidate decides whether it should become a leader. A leader candidate can reach $P_i = 1$ without any member, this means either no other leader agent is willing to accept this agent or all its potential members already join other leaders. In this case, this leader sends join request to all of its known leaders to see if any leader will accept its traffic at this phase. If it still can not get accepted, it promotes itself as an isolate leader. The algorithm for agent member determination can be found in [2].

3.4 Resilience of The Agent Hierarchical Architecture

Resilience to Message Loss. The message exchange between agents in our distributed approach is through multicast. Multicast is not a reliable protocol, packets can be lost during the operation of our algorithm. But due to the periodically sending of search message, leader candidate and leader announcement, this approach can tolerate message loss and still construct the complete agent architecture. The search message loss of an unstructured agent can be compensated by the next round message. The unstructured agent will keep sending till it is covered by a leader candidate or reach maximal round R_T. The loss of leader candidate announcement only causes the unstructured agents sending more searching messages. The loss of leader announcement can cause unstructured agents join suboptimal leader. In extreme case where all leader announcements to an agent get lost, that agent can still send join request during the member determination phase or become an isolate leader which can join higher level or directly report to the manager. So, the packet loss can influence the result architecture, make some agents join suboptimal leaders. But our algorithm guarantees that each agent will find a leader or become a leader, no agent will be isolated from the resulting architecture, this will complete the architecture.

Resilience to Nodes Failure and Topology Change. When network topology changes or agents fail to perform its monitoring tasks, the hierarchical architecture should be reconstructed to reflect these changes. In our proposed algorithm, the distributed leader selection and assignment component contributes significantly to the resiliency of the monitoring infrastructure as the topology changes can be accommodated locally in the affected areas without globally impacting the rest of the architecture. Each leader monitors the topology change in the area it controls and keeps the statistical record of its members' average communication cost. If an abnormal average communication cost lasts for more than a certain period, the leader can assume topology change happens in the network. Then the leader sends out a reconstruction notification message to its members to trigger the selection of new leader. The reconstruction only happens within the scope of old leader and its members such that other areas are not effected. As for agent failure, the normal agent failure will not influence the monitoring architecture. The architecture only need be changed when a leader

node fails. Since each leader node periodically sends out beacon message to its members, when the member agents lose contact from their leader for certain time, they set its status back to unstructured and restart the leader selection. The new selected leader promotes itself to higher level to receive leader beacon messages at that level, so it can select the best leader to join at that level.

3.5 Dynamic Parameters Selection

The proposed distributed algorithm is significantly impacted by two important thresholds: leader candidate threshold P_T and competition relation threshold L_T. In order to make this approach suitable for different networks and agent distributions, these parameters should be dynamically chosen to reflect the target system properties. Since the construction of agent hierarchy structure is totally distributed without any central node assistance, and there is no agent has global knowledge, we propose the following technique for parameters selection. At the beginning of architecture construction, each agent randomly send out a multicast search message ($TTL = 255$) with probability p, which is a small value to limit only a small fraction of agent sending out messages. Upon receives these messages, each agent can calculates the distance to other agents and sends these information to the agent with smallest ID. So the agent with smallest ID has a sample of global knowledge. It then calculate the mean and standard deviation of agent capability, distance, and event rate for the sample agents set. Based on these information, it estimate how many search rounds needed for an agent with average capability and average distance to make its P_i value reach 1. Because the agents need some time to resolve the competition between leader candidates, so the P_T value is set as P_i value of the estimated agent at third last round. And the L_T value is set as the estimated L_{ij} value between two leader candidates share half of their members with average distance. The details of how these thresholds are calculated are shown in [2]. After calculation, the agent with smallest ID will multicast these thresholds to all other agents.

4 Evaluation

In this section, we evaluate the performance of our distributed agents hierarchical architecture construction algorithm using networks with various topologies and sizes. Our simulation focuses on studying the accuracy, scalability and efficiency of our heuristic distributed approach to the LSAP problem and compare the results with the optimal centralized approximation algorithm [12].

Because the target of this approach is enterprise network, so we assume that the network links for each agent have enough capacity to accommodate the event traffics to and from that agent. Agents have different processing capabilities and event traffic rates. The distances between agents are network distances. We assume no path restriction in our model, which implies that each agent can send a message to any other agent. Based on these assumptions, we construct many network topologies for our evaluation study by distributing agents randomly into a two-dimensional graph. The geometry distance between agents symbolize the

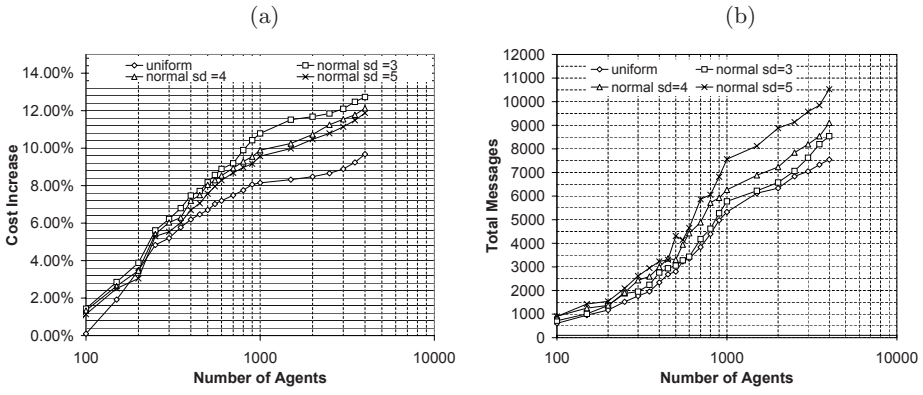

Fig. 1. Performance evaluation (a) Total cost increase ratio (b) Total messages change as agents number changes

network distance. The coordinates of agents are generated with uniform and normal distribution. We use the following performance metrics in our study: (1) total transferring cost which is the summation of the communication cost for each agent,(2) total messages which is the summation of the messages used between agents during the construction,(3) finishing round which represents how long it will take for each agent to finalize its status.

4.1 Accuracy

Recall that the objective of our distributed agents structure construction algorithm in each layer is to find the best subset of agents as leaders and assign member agents to these leaders to achieve minimal total communication cost. We apply our algorithm to several network topologies with agents number varies from 80 to 4000 and compare the total transferring cost ($Cost_{dist}$) with the result of the centralized local search approximation algorithm ($Cost_{cent}$). We use the increase ratio (IR) represent the difference between these two results which is calculated as follows: $IR = (Cost_{dist} - cost_{cent})/Cost_{cent}$. The IR value for different network topologies is shown is in Fig. 1.a. From this graph, we can see that IR value increases slowly as agents number increases dramatically. Our approach uses the hop counts as the distance between agents/nodes, which is obtained by rounding up the geometric distance divided by the predefined distance of one hop. This may lose precision when compared with the centralized approach that uses network distance without rounding. From this figure we can see that, for network generated based on normal distribution, higher standard deviation network gets lower IR value in our approach. This means our approach can achieve better results when agents are widely distributed.

4.2 Scalability

Scalability is another very important criteria to evaluate our distributed agents structure construction algorithm. We want to prevent the search messages from

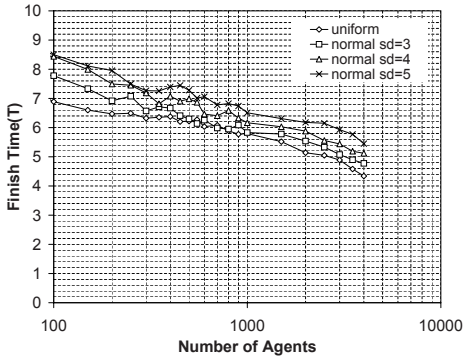

Fig. 2. Average finish time changes as agents number changes

flooding the network. Our objective is to use the minimal number of messages to finish the construction. We evaluate the scalability of our work by applying it on different network topologies and calculate the total messages used for the construction. The result is shown in Fig. 1.b. From this graph we can observe that the total number of messages increases almost linearly as the number of agents increases, make our approach suitable to large-scale networks.

4.3 Efficiency

How fast the agent architecture can be built is crucial to the performance of the distributed monitoring system. Slow construction implies slow detection and reaction to network events. In each level of the agent hierarchy, each agent finishes the construction of the current level (finalize its status) when it becomes a leader or a member of another leader. So different agents might finish the construction at different search round. We evaluate the efficiency of our approach by studying the average finishing time of agents in different network topologies. We set the threshold $R_T = 20$ to terminate the construction for some agents with small processing power and far from rest agents. Fig. 2 shows the average finishing round of agents at different network topologies. We can see that the finish time keep in certain range regardless the network topology changes. Also, as the number of agents increases, each agent receives more search messages during each round. So the P_i value will also increase faster. This is why we can see that as the number of agents increases, the average finishing time decreases.

5 Related Work

To our best knowledge, there is no prior work which addresses the problem of how to construct the agent architecture in a complete distributed and autonomous fashion in large-scale enterprise network based on network topology and nodes capacity. Our related work study will focus on those works address the agents

structure of large-scale monitoring and management system. HiFi [1] is a hierarchical events monitoring systems for large-scale network. In HiFi, the monitoring agent architecture has to be manually configured, which limits its fast deployment in large-scale network. Grid resource and application monitoring tools have been proposed in [4] and [10]. Although these system adopt hierarchical architecture, none of these architectures consider the underlying network topology, nodes traffic and capability.

A topology-aware overlay construction technique was proposed in [11]. In this approach, every overlay node will independently pings a set of predefined landmarks. Each node will sort the set of landmarks Based on the Round_Trip Time (RTT) to them. Thus, each node will have a landmark order vector, which reflects the topology position of that node. All nodes with same landmark order will independently join the same group. This approach can cluster large amount of agents in distributed fashion, but the accuracy of the result depends on the selection of landmarks. It can only sparsely cluster the nodes, and it may create nodes unevenly distribution problem. LEACH [9] proposes a distributed cluster algorithm for sensor networks. Nodes in LEACH make autonomous decisions to become cluster head based on energy consumption without any centralized control. But LEACH doesn't guarantee good head selection and distribution, its objective is to evenly distribute the energy load among all the nodes. On the contrary, in our approach, agents compete with each other to become leaders.

6 Conclusion and Future Works

In this paper, we have addressed the problem of automatic construction of topology-aware hierarchical agent architecture for large-scale event monitoring and correlation systems. Our distributed approach overcomes the shortcomings of centralized approaches and provides a scalable mechanism that can accommodate topology changes. The agent architecture is built in a bottom-up fashion and no central manager is required to organized the agents. We show how to build an optimal architecture based on the network topology and agents' capabilities. The possibility of one agent to become a leader in the hierarchy is determined by its processing power and proximity to other agents. In our approach, the agents compete with each other to become leaders but our algorithm favors the agents with high processing power and more close agents. The simulation results show that this approach can construct the agent architecture efficiently with reasonable communication cost for a large distributed monitoring system.

Our future work will focus on two tasks. First, we will improve dynamic parameters selection algorithm, analyze the influence of different threshold calculation methods and sample size to the accuracy of our approach. Also, we will study the impact of the proposed recovery and reconstruction technique on the optimality of the agent architecture.

References

1. Al-Shaer, E., Abdel-Wahab, H., Maly, K.: HiFi: A New Monitoring Architecture for Distributed System Management. In: ICDCS 1999. Proceedings of International Conference on Distributed Computing Systems, Austin, TX, pp. 171–178 (May 1999)
2. Zhang, B., Al-Shaer, E.: Self-Organizing Monitoring Agents for Hierarchical Monitoring Architecture. Technical Report, multimedia research lab, Depaul University (2007)
3. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems (TOCS) 19(3) (August 2001)
4. Baker, M., Smith, G.: GridRM: An Extensible Resource Monitoring System. In: CLUSTER 2003. Proceedings of the 5th IEEE Cluster Computing Conference, Hong Kong (December 2003)
5. Fei, A., Pei, G., Liu, R., Zhang, L.: Measurements on Delay and Hop-Count of the Internet. In: Proc. IEEE GLOBECOM 1998 Internet Mini-Conf., IEEE Computer Society Press, Los Alamitos (1998)
6. Gruber, R.E.: Balachander Krishnamurthy and Euthimios Panagos: High-level constructs in the READY event notification system. In: Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications, Sintra, Portugal (1998)
7. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Transactions on Computer Systems (TOCS) 19(3) (August 2001)
8. Gruber, R.E., Krishnamurthy, B., Panagos, E.: High-level constructs in the READY event notification system. In: Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications, Sintra, Portugal (1998)
9. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: An Application-Specific Protocol Architecture for Wireless Microsensor Networks. IEEE Transactions on Wireless Communications 1(4), 660C670 (2002)
10. Truong, H.-L., Fahringer, T.: SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid. Technical report, Institute for Software Science, University of Vienna (October 2003)
11. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Topologically-aware Overlay Construction and Server Selection. In: INFOCOM (2002)
12. Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a Local Search Heuristic for Facility Location Problems. In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1–10. ACM Press, New York (1998)
13. Osman, I.H., Christofides, N.: Capacitated clustering problems by hybrid simulated annealing and tabu search. Transactions in Operational Research 1, 317–336 (1994)
14. Jain, K., Vazirani, V.: Primal-dual approximation algorithms for metric facility location and k-median problems. In: Proceeding of the 40th Annual IEEE Symposium on Foundation of Computer Science, pp. 1–10. IEEE Computer Society Press, Los Alamitos (1999)
15. Yemini, S.A., Kliger, S., Mozes, E., Yemini, Y., Ohsie, D.: High Speed and Robust Event Correlation. IEEE Communication Magazine, 433–450 (May 1996)