A Model for Adaptive Monitoring Configurations

H. Abdu, H. Lutfiyya, M. Bauer
Department of Computer Science
The University of Western Ontario
London, Ontario, Canada
{hasina, hanan, bauer}@csd.uwo.ca

Abstract

With the increased availability and complexity of distributed systems comes a greater need for solutions to assist in the management of distributed system components. Despite the significant contributions made towards the development of management tools that monitor and control distributed systems, little has been done to address issues such as the cost of management and how it can adapt to the dynamic changes in user requirements as well as system resources.

We present an adaptive model, in which an initial optimal configuration of management agents is determined according to a set of user/system requirements. These agents can later be dynamically reconfigured to adapt to changes in resource availability and user/system constraints, with minimal effect on the behaviour of the managed system components. Algorithm, prototype, and experimental results are presented.

Keywords

Distributed Systems Management, Monitoring Overhead, Management Agents, Monitoring Directives

1. Introduction

With the increased availability and complexity of distributed systems comes a greater need for solutions to assist in the management of system components (e.g. resources, networks, application processes). The management environment needs to collect data about the existence and behaviour of services and devices of many kinds. The data has to be kept for analysis based on different viewpoints, such as the performance of a particular service, or the availability of a set of services, and for analysis over different time scales. The management system also has to provide automatic reactions of various kinds to maintain services and service quality. The collection and control functions are performed by entities referred to as **management agents**. Agents communicate with managed objects to collect information relevant for management.

Large computer systems are expected to have a long lifetime. The services it is expected to perform will evolve as the needs of users evolve. This means that a large system cannot be assumed to remain static for its operational lifetime. Operational changes, such as physical/logical location, component degradation, system re-dimensioning etc., need to be catered for. This, in turn, implies that management needs will change. Thus, the set of agents being used need to be dynamic to accommodate the changes. Even without changes to the system, management requirements may change as a function of changes in user expectations of the behaviour of the system.

While significant contributions have been made towards the development of management tools that monitor and control distributed system components, little has been done to address issues such as the cost of management and how it can adapt to the dynamic changes in user requirements as well as system resources. Current research examines how to facilitate agent reconfiguration. For example, management by delegation ([1]) allows tasks to be specified by packing into a program a set of commands and sending this to the appropriate agent. This is a precursor towards the work in code mobility which has been raised mainly by a new family of programming languages, usually referred to as **mobile code languages**. This allows a client to pass code to a server to be executed on the machine that the server is on ([2, 3]). However, little of the current research examines this problem from the following view: Agents consume the same resources used by managed objects. Thus, an overhead in the managed system is unavoidable. The question is the following: Can it be minimized?

Our work focusses on the problem of minimizing overhead due to monitoring. Monitoring involves the collection and analysis of information concerning the behaviour of managed objects. We refer to the specification of the set of attributes and operations, representing the data obtained from managed objects, and the analysis executed on this information, as **monitoring directives**. Our definition of monitoring directives include synchronous polling done by management applications, as well as asynchronous events sent by managed objects to the management application.

By specifying data and operations, directives determine the type of agents to be allocated (i.e. agents can collect different types of data, can execute different operations, run under different protocols, etc.). There are, however, questions such as: How many agents are to be allocated? Where should they be started? Which agents collect data from which managed objects? Will the operations be executed centrally by a single agent, or will they be distributed among various agents? In the latter case, how will the various agents communicate? Is there any specific topology to be followed? Will there be enough resources to allocate all these agents? If so, how should the resources be distributed? Is there any specific resource whose usage should be minimized? These and other constraints are used to model an 'arrangement' of agents and managed objects which we refer to as **monitoring configuration**. An efficient monitoring configuration, that satisfies all user/system requirements (referred to as the *optimal configuration*), is the key to minimize the cost of monitoring, by efficiently allocating resources from the monitored distributed system.

We propose a model to optimize the monitoring of distributed systems, by determining the optimal configuration that minimizes monitoring overhead according to a set of user/system requirements. The optimal configuration can further be modified as changes occur in the system, with minimal overhead on the monitored system.

The rest of this paper is organized as follows: Section 2 describes our model. Section 3 illustrates the implementation of the model. Section 4 describes our prototype. Validation of the model through experiments is described in Section 5. Summary and concluding remarks are discussed in Section 6.

2. Model

Optimizing monitoring involves minimizing overhead, based on a set of constraints. The monitoring overhead of a set of agents is reflected by the consumption of resources such as CPU, memory and network traffic (referred to as *overhead metrics*).

In this section, we define a monitoring configuration as a directed graph, with costs associated with its nodes and edges. These costs are a measure of different overhead metrics. This motivates the formulation of an Integer Linear Programming (ILP) [4] model, in which the variables are nodes and edges of the monitoring configuration graph, the constraints are user and system requirements, and the objective function coefficients are the cost of overhead metrics.

2.1 Definitions

Definition 2.1.1. A *monitoring configuration* is a directed graph $MC = (V, E)$ where $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of nodes, and E represents the data flow between nodes in V. There is an outgoing edge from node v_i to node v_j, represented by $e_{i,j} = (v_i, v_j)$, if and only if v_i requests information from v_j, or v_j asynchronously sends data to v_i. For example, if v_i is an agent collecting information from a managed object represented by node v_j, then there is an outgoing edge, $e_{i,j}$, from v_i to v_j. Similarly, if v_j is a process that notifies v_i that it has timed out too many times, then there is an edge $e_{i,j}$ from v_i to v_j. □

The nodes of a configuration graph represent management agents and managed objects. Agents can be of two types: *collector* and *analysis*. Collector agents collect information from managed objects and transmit the data to analysis agents, responsible for performing analysis operations on the collected information. Operations can be executed by a single analysis agent, or can be distributed among a set of communicating analysis agents, depending on the defined constraints. Figure 1 illustrates possible monitoring configurations for a sample directive (for purpose of illustration, we use a simple example). The directive specifies an *average* operation on the memory usage collected from six managed objects. In Figure 1(a) all the data is collected and analyzed by a single agent, whereas, in (b), data is collected by six different collector agents, and the analysis is distributed among different analysis agents.

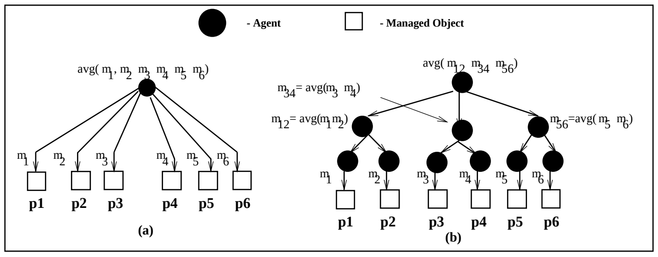


Figure 1: Sample monitoring configurations

The cost associated with a configuration graph is the sum of the costs associated with its nodes and edges.

Definition 2.1.2. The cost of a monitoring configuration MC is defined as follows:

$cost(MC) = \sum_{v_i \in V} cost(v_i) + \sum_{e_{i,j} \in E} cost(e_{i,j})$ (1) □

The costs associated with the nodes and edges of a graph are defined as follows:

Definition 2.1.3. The cost of a node v_i is defined as $cost(v_i) = c_1(v_i) + c_2(v_i) + \ldots + c_n(v_i)$, where $c_1, c_2, \ldots, c_n$ are overhead metrics, such as CPU (number of processes in queue), memory (amount of memory in use by a process while the process is running) and response time (time between an operation invocation made to a process, and the completion of that operation), and depend on the type of agents and managed objects being represented, the rate in which data is being collected, and the type of analysis being done. □

Definition 2.1.4. The cost of an edge $e_{i,j} = (v_i, v_j)$ is defined as $c(e_{i,j})$ which represents the network traffic between nodes v_i and v_j. Network traffic is generated only between nodes located in different hosts, in which case it is measured by the number of packets transmitted during monitoring. This traffic depends on factors such as the directive being evaluated (e.g. polling frequency, size of the collected information type, sampling period, etc.), the location of nodes, and the type of agents involved (i.e. agents differ in protocols they use: some are connection-oriented and others are connection-less, thus differing in the generated network traffic).

We define the cost of an edge $e_{i,j}$ in function of the number of bytes required to send one packet of data. This value will depend on factors such as the protocol used by managed objects and agents. For example, CMIP agents require the exchange of more control bytes compared to SNMP agents. If n is the size of the attribute being collected, and c is the number of control bytes required for sending a packet through an edge, the total number of bytes required for each time data is sent is a function of $(c + n)$. □

There is a clear trade-off between the different costs defined above. For example, having multiple agents and managed objects allocated in one host reduce network traffic. However, the memory usage and CPU load on that host would increase. Thus we specify a subset of the costs to be minimized that, along with other user/system constraints, is included in Req, the set of user/system requirements to be satisfied when optimizing the monitoring of a distributed system. Examples of requirements include the following:

- Resource constraints: specify what resource is to be minimized (e.g. network traffic, response time);
- Topology constraints: specify preferred topology of agents (e.g. centralized, binary-tree configuration, etc.);
- Configuration constraints: specify preferences on the number and location of agents (e.g. only two agents per host, collector agents may only run on host *spud.csd.uwo.ca*);
- Directive constraints: specify the directives to be evaluated (e.g. attributes, operations, polling frequency, etc.).

We can now define the problem of optimizing monitoring as follows:

Definition 2.1.5. *Monitoring optimization* consists of determining a configuration MC of agents and managed objects that satisfies a set of user/system requirements (Req), given a set of monitoring directives (D), managed objects (M), and management agents (A); and that can be reconfigured upon changes in Req, with minimal effect on M. □

2.2 Modelling Adaptiveness

In this section, we model the monitoring optimization problem as a 0-1 Integer Linear Programming (ILP) [4] problem, focussing on enabling reconfigurable monitoring configurations. Figure 2 illustrates the structure that will be used by our model.

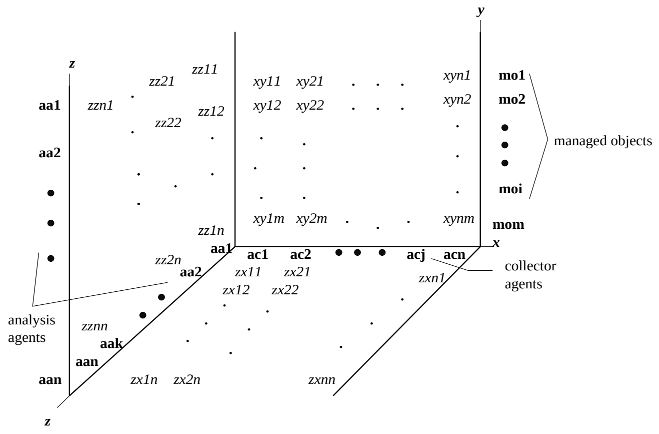


Figure 2: Structuring the optimization problem as an ILP problem

Collector agents are represented by x, managed objects by y, and analysis agents by z. Collector agents collect information from managed objects. Information is then passed to one or more analysis agents, that execute analysis operations on the collected data. These interactions are represented by the variables xy, zx and zz.

A variable xy_{ij} in Figure 2 represents the communication between the j^{th} managed object and the i^{th} collector agent. A variable zx_{ij} represents the communication between the i^{th} analysis agent and the j^{th} collector agent, and a variable zz_{ij} represents the information passed from the i^{th} analysis agent to the j^{th} analysis agent. These variables can be either 0 or 1, representing the absence or existence of an edge between corresponding nodes [5].

Finding the optimal monitoring configuration consists of determining the values of the variables xy, zx and zz (the edges of the monitoring configuration) in such a way that the requirements specified in Req are satisfied, and the associated cost of the configuration is minimal. When formulating this problem as an ILP problem, we have:

$$\begin{array}{ll} \text{minimize} & f = CX = \sum_{ij} cy_{ij} xy_{ij} + \sum_{ij} cx_{ij} zx_{ij} + \sum_{ij} cz_{ij} zz_{ij} \quad (2) \\ \text{subject to} & AX \leq B, \quad X = \{0,1\} \end{array}$$

Where X is the n-dimensional vector of variables, representing the xy, zx and zz variables. C is the n-dimensional vector representing the cost of the n variables in X, according to our definition of monitoring configuration costs (in this example, the values in C represent the network traffic cost between nodes). A is the $m \times n$ matrix and B is the m-dimensional vector of constraint coefficients, representing the m constraints obtained by translating the requirements in *Req* into mathematical equations (examples in Section 4.).

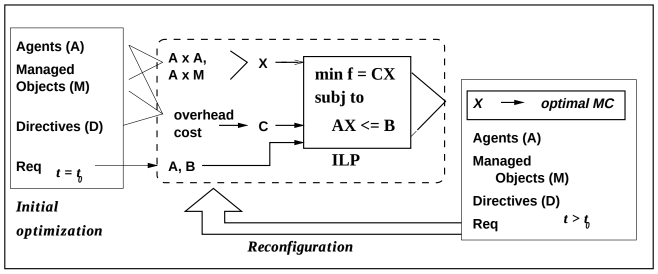


Figure 3: A model for adaptiveness

Figure 3 illustrates our model from an operational view. At time $t = t_0$, agents (A), managed objects (M), directives (D) and a set of requirements *Req* are mapped into the ILP function coefficients (C), variables (X), and constraint matrices (A and B). The value of X that minimizes f, satisfying $AX \leq B$, is determined, representing the edges of the optimal monitoring configuration MC.

If, at any time $t > t_0$, agents, managed objects, directives or the optimization constraints are modified, MC is dynamically reconfigured. The new input to the algorithm includes the existing optimal solution, thus saving computation. Reconfiguration is further explained in the next section.

3. Algorithm

We use a modified version of the implicit enumeration algorithm [6] to implement our optimization problem. In this method, all solutions are enumerated but the vast majority are enumerated *implicitly*. Only a few are *explicitly* enumerated.

In an n-variable 0,1 problem there will be 2^n possible solutions, most of which may be infeasible. Instead of generating all the 2^n solutions and verifying the feasibility of each one of them, the implicit method starts with one of the 2^n solutions. If there are constraints violated by this initial solution (constraints in $Ax \leq B$), the algorithm selects a set of candidate variables that, if raised to 1, can bring about feasibility. A variable can be a candidate if it has a negative coefficient in matrix A (due to the '$\leq$' sign), and a cost c that, when added to the current value of f will not increase it more than a specified limit (the new variable, when added to the solution, must not result in a value of f that is greater than the current feasible solution). If there is more than one candidate, the one with most negative coefficient in A is chosen. The variable is added to the current partial solution S, f is updated by adding the coefficient of the new variable, and the value of the variable is set to 1.

The above process is continued until a partial solution is fathomed, either by feasibility or in-feasibility. 'Fathoming' occurs once a partial solution that cannot be completed in such a way as to avoid violating one or more of the constraints is found. For example, in a 4 variable (x_1, x_2, x_3, x_4) problem, if the combination (0, 0, 1, 0) is tested, and it is verified that none of the constraints can be satisfied with $x_3 = 1$, then all the 8 solutions (2^3) having $x_3 = 1$ have been implicitly enumerated and can be ignored, thus being fathomed. Similarly, once we find a feasible completion of a partial solution we say that the partial solution is fathomed because no other completion of it is more attractive than that completion in which all other variables are equal to zero. In other words, all combinations where the current variables in the solution are equal to one have been implicitly fathomed. Thus, in a problem with 5 variables, if 2 variables are set to 1 in the feasible solution, then $2^{(5-2)}$ possible solution have been implicitly fathomed.

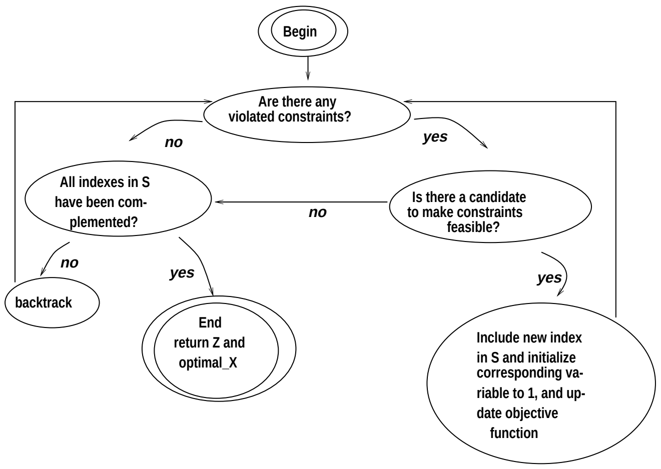

Begin
Are there any violated constraints?
no
yes
All indexes in S have been com-plemented?
no
Is there a candidate to make constraints feasible?
no
yes
yes
backtrack
End return Z and optimal_X
Include new index in S and initialize corresponding va-riable to 1, and up-date objective function

Figure 4: State machine for the implicit enumeration algorithm

Once a solution is fathomed (due to feasibility or in-feasibility), the algorithm 'backtracks'. Backtracking consists of fathoming the existing solution, i.e. searching for a better solution, in which the last variable included in the current partial solution (x_i) is set to zero. This way all solutions are enumerated, since the solutions in which x_i is set to 1 were already fathomed before backtracking.

The algorithm ends when a new feasible solution is found during a backtracking due to feasibility, or if no more variables can be set to zero during backtracking. In the latter situation, the algorithm returns the last existing feasible solution (Z). At any iteration of the algorithm, it is obvious by inspection of the current partial solution exactly how many and which solutions have been implicitly enumerated so far. It is clear that every time fathoming occurs, 2^{n-1} solutions have been implicitly enumerated. Thus, an optimal solution can be found in very few steps. Figure 4 illustrates the main steps of this algorithm.

We use the 'backtracking' feature of the implicit enumeration method to incorporate the dynamic reconfiguration capability into our model. For example, if an initial optimal configuration is determined at time $t = t_0$, and, at time $t = t_n$, an agent has to be migrated to another location, the optimal configuration vector (X) is reused in such a way that the backtracking is started on the variables that correspond to the agent being migrated. This way, all the other edges, that may not have to be modified, do not have to be recomputed.

In other situations, the existing solution vector is modified according to changes occurred in the system, in order to provide an initial feasible solution closer to optimality. For example, if new managed objects are to be monitored, the variables corresponding to the new edges are added to the model. Instead of initializing them to zero, the algorithm initializes to 1 the edges between these managed objects and one of the existing agents. The ability to utilize existing feasible solutions is one of the main advantages of this algorithm.

There are no specific numbers to represent the amount of computation saved during reconfiguration. This depends on the type of changes occurred in Req, and on the existing monitoring configuration. For example, the amount of computation saved during reconfiguration due to changes in configuration constraints (e.g. location of agents and managed objects) is greater than what is saved when changing topology constraints.

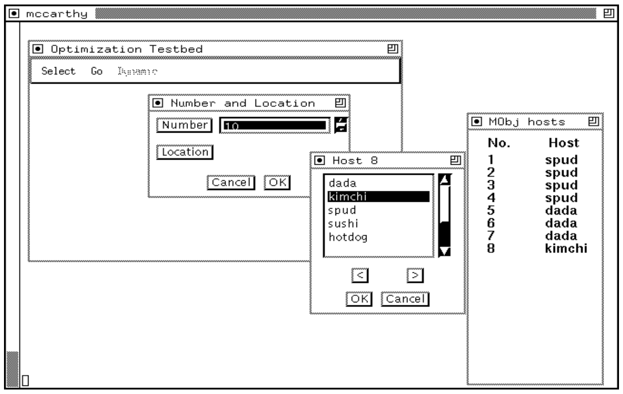
4. Prototype

Our prototype consists of an implementation of the algorithm and a Java application that enables users to specify the number and location of managed objects and agents (see Figure 5). Users can also set constraints to the problem, by selecting requirements, which are mapped to mathematical constraints. For example, if in Figure 5 the user selects 10 managed objects, 3 collector agents and 3 analysis agents in different locations, we have the structure showed in Figure 6 (the names beside managed objects and agents indicate their location specified by the user), with a total of 48 $((10 \times 3) + (3 \times 3) + (3 \times 3))$ variables, resulting in 2^{48} different monitoring configurations.

The user can now select the constraints to the problem. For example, in Figure 7, the user is given the option to select the overhead metric to be minimized. The selected constraints are mapped to equations. For example, a resource constraint that minimizes network traffic results in having the objective function coefficients representing the network traffic between nodes. Topological constraint opting for a hierarchical structure (i.e. only one incoming edge per node) would result in the following constraints:

- $xy_{11} + xy_{21} + xy_{31} \leq 1$: not more than one edge from collector agents to mo_1
- ...
- $xy_{110} + xy_{210} + xy_{310} \leq 1$: not more than one edge from collector agents to mo_{10}
- $zx_{11} + zx_{12} + zx_{13} \leq 1$: not more than one edge from analysis agents to ac_1
- $zx_{21} + zx_{22} + zx_{23} \leq 1$: not more than one edge from analysis agents to ac_2
- $zx_{31} + zx_{32} + zx_{33} \leq 1$: not more than one edge from analysis agents to ac_3
- $zz_{11} + zz_{12} + zz_{13} \leq 1$: not more than one edge from analysis agents to aa_1
- $zz_{21} + zz_{22} + zz_{23} \leq 1$: not more than one edge from analysis agents to aa_2

- $zz_{31} + zz_{32} + zz_{33} \leq 1$: not more than one edge from analysis agents to aa_3.



No.	Host
1	spud
2	spud
3	spud
4	spud
5	dada
6	dada
7	dada
8	kimchi

Figure 5: Choosing location of managed objects

All constraints are mapped into matrix A and vector B, which are used by the ILP algorithm to find the optimal configuration (Figure 8).

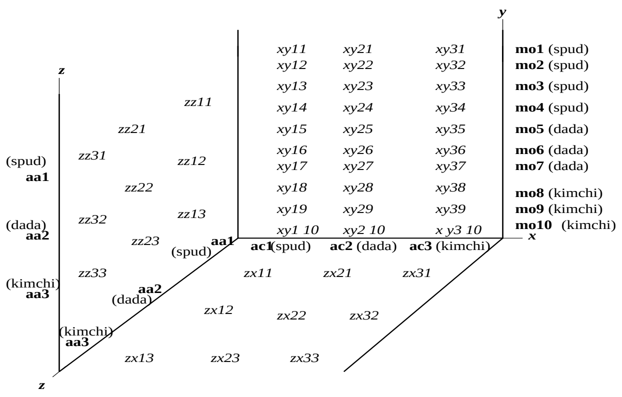


Figure 6: A sample input for the optimization algorithm

The user can now dynamically change the input to the algorithm. Figure 9 illustrates some options.

The next section describes some of the experiments performed to validate our model, along with results that compare the iterations taken by the algorithm to determine the initial configuration and to perform reconfiguration.

5. Experiment

We have a set of experiments to test the effectiveness of the formulated problem and algorithm. We compared the overhead of our 'optimal' configuration with the overhead of other possible configurations, in terms of the resource being minimized. We will present the results of one of our experiments.

Our monitoring system is based on the OSI Management Framework, where the CMIP protocol is used for agent-to-agent communications. We have two

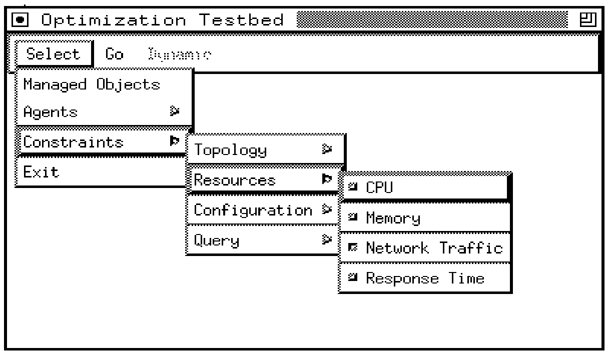


Figure 7: Resource constraints

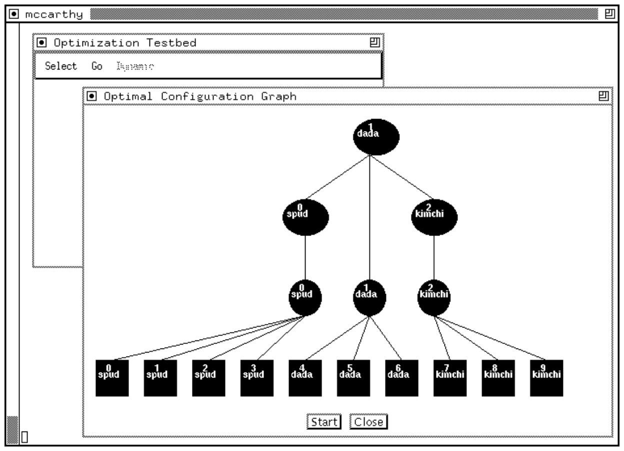


Figure 8: The optimal configuration

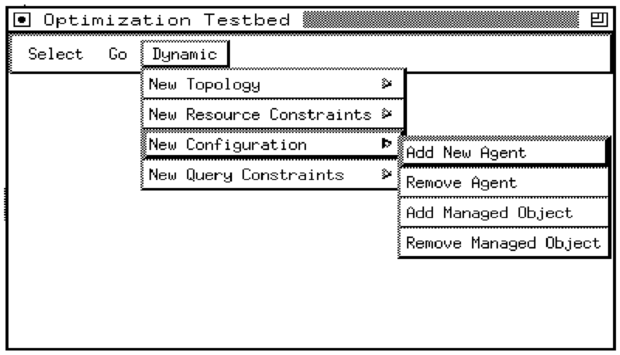


Figure 9: Changing requirements

types of agents: a DCE Collector Agent and Statistics Agent. The DCE Agent is used to collect information from DCE application processes (we use a sample OSF DCE based distributed *Linear Programming* application). It has a CMIP Interface to communicate with the OSI environment and a DCE Knowledge Source that enables the use of DCE RPC to communicate with DCE application processes. The Statistics Agent analyzes the data collected by DCE agents. Currently, statistics agents support four operations: max, min, avg, and total. Agents and managed objects (DCE application processes) can run in three $RS/6000$ machines (dada, kimchi and spud). These are configuration constraints set by the system. These agents, managed objects and the set of directives and user/system requirements were the input to the ILP model. The directives and requirements are as follows:

- Directives:
  - Average number of messages sent by all managed objects;
  - Sum of the process identifier of all managed objects;
- Requirements:
  - Resource constraint: minimize network traffic;
  - Configuration constraint: one collector and one analysis agent per host;
  - Topology constraint: every node has a maximum of one incoming edge;
  - Directive
    - attributes: number of messages, PID
    - operations: average and total.

The directives were deliberately kept simple for illustration purpose. The resource constraint determines the costs being minimized. In this example, the coefficients of the objective function ($f = CX$) represent the network traffic between agents, and between an agent and a managed object. This coefficients were determined based on the agent and managed object frameworks (e.g. OSI, DCE).

We compared the network traffic generated by the 'optimal' configuration with the network traffic generated by three other configurations: single agent, 'expensive', and binary configuration. The comparison was repeated for different number of managed objects (i.e. $2, 4, 6, 8$ and 10), resulting in 20 different monitoring configurations. In the single agent configuration, one centralized agent collects information from all the managed objects. In the 'expensive' configuration, all the edges in the graph are between agents and managed objects in different hosts. The binary configuration has the feature of limiting the number of outgoing edges from an analysis agent to a maximum of two.

For example, the 4 configurations for 10 managed objects are illustrated in Figure 10. The main difference between (b) and (d) is that, in (d), the managed objects located in a given host send their information to a collector agent located in a remote host, as opposed to an agent in the same host (as in (b)).

The network traffic produced by hosts was measured using the UNIX *netstats* utility. The comparison was done using the number of packets remotely transmitted during monitoring. Results are shown in Figure 11.

The graph also shows that the network traffic was affected by simply switching the location of agents from the *'optimal'* configuration in Figure 10(b) to the *'expensive'* configuration in (c).

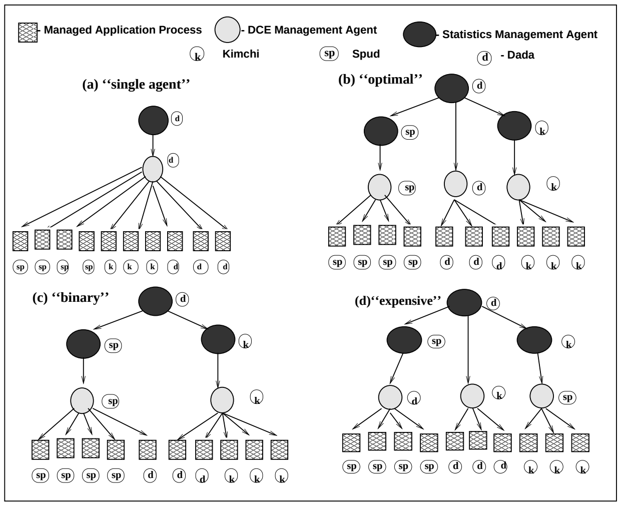
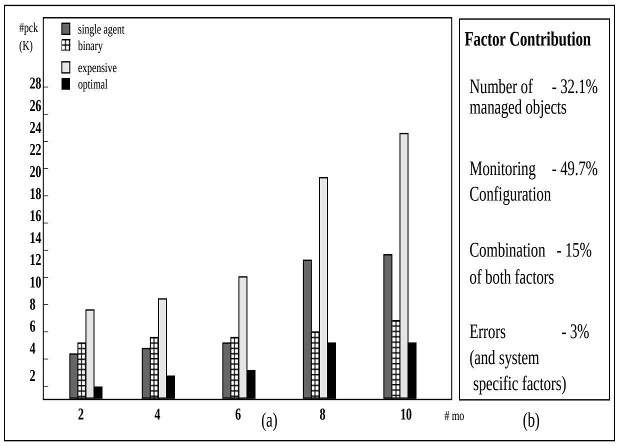


Figure 10: Different configurations with ten managed objects



Factor Contribution	
Number of managed objects	- 32.1%
Monitoring Configuration	- 49.7%
Combination of both factors	- 15%
Errors (and system specific factors)	- 3%

Figure 11: Network traffic generated by monitoring

The measured network traffic was not only due to monitoring, but also due to the monitored application itself, and other activities in the system. To verify the actual contribution of the monitoring configurations to the monitoring overhead, we performed a factorial analysis on the data. The two main factors were: number of managed objects and monitoring configuration. The results are shown in 11(b). The percentage of errors represents fluctuations in network traffic, and contribution from other activities in the system. Due to the dynamic nature of distributed systems, there are constant fluctuations in network traffic values and other resources, resulting in outliers that affect measurements.

We also observed a trade-off between network traffic and other resource usage, such as the amount of memory used by agents. This confirms the fact that monitoring overhead involves different metrics, and the choice of what resource is to be minimized is a trade-off decision that depends on user and system constraints.

In the second part of the experiment, we verified the efficiency of the algorithm, in terms of reconfiguration. This was done by changing constraints dynamically, and comparing the computation required to find the new optimal solution, with the computation required for the optimization done 'from scratch', i.e. without using the existing optimal solution. For a given number of variables, after the initial optimal configuration was determined, reconfiguration was repeated by modifying one constraint at a time. An average of the computation saved for each reconfiguration was taken, representing the percentage of computation saved by reconfiguration, for a given number of variables. This result is shown in Figure 12.

#variables	%saved
15	70%
19	53%
25	40%
41	50%
56	80%
111	80%

Figure 12: Efficiency of reconfiguration

6. Concluding Remarks

We motivated the need of an adaptive model in which management agents and managed objects can be reconfigured according to dynamic changes in management requirements, user/system constraints and resource availability, in such a way that resources are efficiently utilized.

We presented an algorithm that determines an optimal configuration for a given set of agents, managed objects, directives and user requirements. The algorithm also determines new configurations according to changes in user/system constraints. This reconfiguration is done using the existing optimal solution, thus reducing computation. The algorithm is part of our optimization testbed, which enables users to specify constraints to the optimization algorithm, such as the resources to be minimized, and the type of configuration to be selected. The algorithm and our assumptions on the effects of monitoring were confirmed through experiments.

We believe that this work is an important contribution towards creating adaptive monitoring systems, with minimal cost on the monitored system. A similar approach can be found in [7], where agents are dynamically reconfigured according to changes that may occur in the system. The reconfiguration is done according to a *cost* function that determines the network traffic generated by agents. The main difference, however, is that [7] focusses on a framework based on mobile agents, thus not applicable to other types of monitoring systems, and with drawbacks such as security.

Despite the positive results, there are important points to be analyzed, and that are part of our future research. These include:

- How accurate are the cost functions assigned to overhead metrics? Are there any efficient techniques that can be used to assign these costs, considering features such as protocol and agent architecture, that very from system to system?
- What is the cost of optimization? How expensive is dynamic reconfiguration? In what systems would this method be more applicable?
- How would reconfiguration be done in the case of multiple directives?
- How would this solution be applied to larger systems? Currently, our work has been applied to relatively small systems. We will examine larger systems where constraints include organizational boundaries or technological constraints in the model.

References

[1] G. Goldszmidt and Y. Yemini. Distributed Management by Delegation. *15th International Conference on Distributed Computing*, 1995.

[2] M. J. Williams and A. T. Bendiab. A Toolset for Architecture Independent, Reconfigurable, Multi-Agent Systems. *First International Workshop, MA '97 - Lecture Notes in Computer Science*, pages 210–221, 1997.

[3] M. Baldi, S. Gai, and G. P. Picco. Exploiting Code Mobility in Decentralized and Flexible Network Management. *First International Workshop, MA '97 - Lecture Notes in Computer Science*, pages 12–26, 1997.

[4] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison Wesley, 1984.

[5] H. Abdu, H. Lutfiyya, and M. Bauer. A Testbed for Optimizing the Monitoring of Distributed Systems. *To appear in Proceedings of PDCS'98*, 1998.

[6] D. R. Plane and C. McMillan. *Discrete Optimization.* Prentice-Hall Inc, 1971.

[7] A. Liotta, G. Knight, and G. Pavlou. Modelling Network and System Monitoring Over the Internet with Mobile Agents. *IEEE/IFIP Network Operations and Management Symposium Conference Proceedings*, pages 303 – 312, 1998.