FACILITATING EFFICIENT AND RELIABLE MONITORING THROUGH HAMSA

David Breitgand,[1], Danny Dolev[1], Danny Raz[2], and Gleb Shaviner[1]
[1]*School of Engineering and Computer Science*
The Hebrew University
Jerusalem 91904, Israel[*]
{davb, dolev, gleb}@cs.huji.ac.il
[2]*Department of Computer Science*
The Technion, Haifa 32000, Israel[†]
danny@cs.technion.ac.il

Abstract: Monitoring is a fundamental building block of any network management system. It is needed to ensure that the network operates within the required parameters, and to account for user activities and resource consumption.

In the SNMP paradigm, network management systems have been structured using a two-tier architecture with managers being *thick* clients, and the target agents being *thin* servers. This architecture may be unreliable in times since it depends on the management station having an access to the targets. Network distance between the manager and the network elements also imposes high overhead traffic, large processing overheads, and long control loops. To overcome these drawbacks, distributed network management architectures based on a middleware layer were proposed. However, such approaches suffer both from the need to modify network elements, and the high (and sometime hard to predict) overhead and complexity.

In this paper we study a solution based on a lightweight middleware architecture that aims primarily at improving availability and efficiency of monitoring applications. We describe the *Highly Available Monitoring Services Architecture (HAMSA)*, present its implementation details, and evaluate its performance. Specifically, we demonstrate how the system can be easily deployed and used for several monitoring applications. HAMSA allows a high level of availability and abstraction, with relatively low overhead.

Keywords: monitoring, network management, high availability, middleware, group communication.

1. Introduction and Motivation

The traditional two-tier structure of network management applications based on the rigid client/server paradigm with thick managers and thin target agents suffers from scalability and availability limitations [16].

Among the more prominent problems with this approach are the following.

[*] This research was supported in part by Intel COMM Grant - Internet Network/Transport Layer & QoS Environment (IXA), and by Check Point PhD Fellowship program, Israel.

[†] Partial funding provided by the Fund for the Promotion of the Research at the Technion, and the Technion V.P.R. Fund - New York Metropolitan research Fund.

- Since the management agents have limited functionality, and only instrument the access to the management data, all the computations should be performed at the manager. Thus, large volumes of data should be transferred over the network, and the traffic overhead can be high.
- As shown in [7] reactive (*i.e.*, event-driven) monitoring schemes are far more efficient in terms of communication than polling-based ones. However, in standard management frameworks, such as SNMP, configuring application-specific threshold-driven traps is a non-trivial task, and not always possible.
- Manager station becomes a processing bottleneck as the size of the managed network increases.
- Manager station is a single point of failure, which hurts availability of the monitoring services. Although for some types of management data disconnected operation of monitoring can be achieved using RMON [15], the disconnected monitoring operation is not available in the general case. This type of operation, however, is essential for scaling monitoring services, reducing communication overhead, and increasing survivability of management services as explained below.
- The unavoidable network distance between the management station and (some of) the network elements makes it very hard to control the elements, due to the inherent instability imposed by long control loops.

Because of these problems, alternative approaches to monitoring architectures are being pursued [5, 6, 11, 12, 4, 13, 14]. The more important of these emerging approaches and their relationship to our proposal are discussed in Section 5.

One alternative is adopting the popular three-tier architecture. A typical three-tier monitoring application is described in Figure 1b. The target agents constitute the lowest tier and serve as the source of the management data. The monitoring components are dynamically dispatched at the middle tier. They monitor the target agents (and, possibly, communicate with other monitoring components) accumulating and pre-processing the information collected from them. The end-consumers of this information, the management front-ends, constitute the uppermost tier.

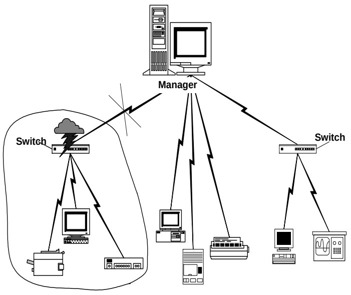


Figure 1a. Two-tier approach.

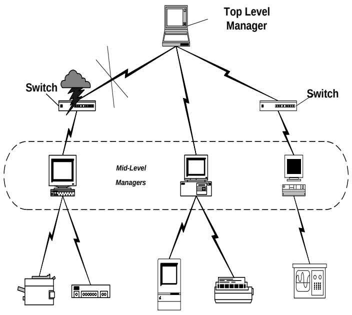


Figure 1b. Three-tier approach

Note that in the three-tier architecture, the components residing in the middle tier can partially or fully implement some of the processing functionality that was previously residing exclusively on the manager side. Thus, using this architecture reduces the traffic overhead, shortens the control loops, and extends the management functionality. In particular, the middle-tier components can implement efficient application-specific event-driven monitoring schemes. Survivability and availability of the network monitoring services are also improved. The mid-tier components can operate autonomously of the first-tier managers (see Figure 1b). When certain parts of the monitored network become unavailable, *e.g.*, due to a network split, the mid-tier monitoring components can continue monitoring in their respective partitions, and later merge the results. This is impossible in the centralized two-tier architecture.

On the down side, the three-tier client/server applications are much more difficult to control. Providing high availability of the mid-tier components in spite of host crashes, network splits and merges is especially challenging.

A standard way of creating a three-tier client/server application is using *application server* that provides the mid-tier run-time environment. However, to the best of our knowledge, no existing application server provides a highly available run-time environment that copes with the kind of failures described above, and can transparently restarting failed *stateful* monitoring components from a consistent point.

Given the complexity of handling distributed three-tier applications in an unpredictable environment prone to various network failures, it is both important and challenging to provide a maximally transparent infrastructure that allows a manager to deploy the needed monitoring components of the second tier in a highly available manner. This improves the overall failure behavior of the management applications, allows more efficient applications (such as event-driven monitoring applications), and therefore contributes to better provisioning of network services in general.

In this paper we propose a novel middleware architecture, HAMSA, that facilitates reliable and efficient deployment of three-tier monitoring applications. We describe the main building blocks of this architecture, and demonstrate it power for efficient and reliable monitoring by describing and analyzing the performance of monitoring applications implemented using HAMSA.

2. Architecture Overview

HAMSA is an architecture that defines the interfaces, and functionality of a highly available run-time environment for HAMSA-Compatible Components. We use the term *components* (as done in many other middleware software systems) to describe objects that implement a set of the predefined interfaces allowing dynamically to mix and match this object with other objects that conform to the same set of interfaces. The following guarantees on the execution of these components in the second tier generate the primary added value of HAMSA.

- Failures of the components are masked from the outside entities. As long as HAMSA has sufficient resources for executing the components, components function *continuously* despite component host failures (*i.e.*, the machine running the component), arbitrary asynchronous network splits, merges, and host recoveries.

- In case of network splits, a single instance of each component is executed per network partition.

- The component whose host machine has failed is guaranteed to resume operation from the last consistent state, *i.e.*, the last state of this component known to the outside world.
- Component's interactions with the environment may potentially influence the state of other components, and/or external entities. In this case, interactions (messages, method invocations) are termed *non-idempotent*. Failure, and a subsequent recovery, of a component being in the middle of a non-idempotent interaction may violate the original interaction semantics. An advantage of HAMSA over other middleware architectures is that it preserves the original *at-most-once*, or *at-least-once* semantic of the component interactions in spite of asynchronous network failures.

HAMSA highly available execution model is not trivial to achieve. Due to the lack of space we do not elaborate on these issues. More details will be provided in the full version of this paper.

These advantages come at a certain price in bandwidth and processing overhead. In Section 4 we discuss the trade-offs between the extended functionality of HAMSA and this overhead. Also HAMSA restricts the inter-process communication model to asynchronous communication and messaging. This communication model, though, fits well into the NM domain.

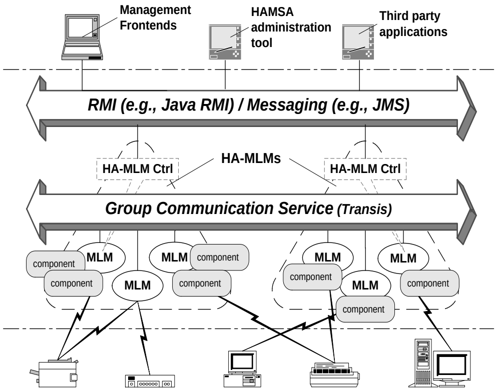


Figure 2a. Detailed Logical Structure of HAMSA

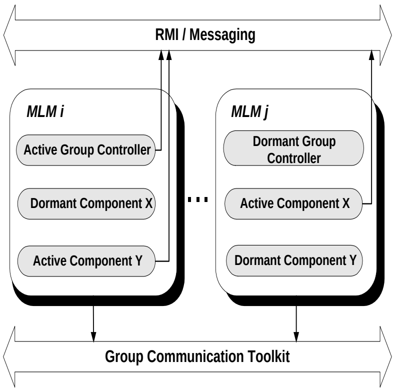


Figure 2b. Logical Structure of HA-MLM

2.1 Highly Available Mid-Level Managers (HA-MLMs)

The run-time environment with the above properties is provided by a set of virtual servers termed *Highly Available Mid-Level Managers (HA-MLMs)*. HA-MLMs are logical entities that are comprised of one or more physical servers called MLMs (see Figure 2a). To its users, every HA-MLM appears as a single object. Its interface is exported by the special component called *HA-MLM Controller*. Each MLM in a given HA-MLM is capable of running the HA-MLM Controller, but at any given moment only a single MLM is executing it. At other MLMs this component is being *dormant*, see Figure 2b.

HAMSA-compatible component is dynamically delegated to a specific HA-MLM through its controller interface using the HAMSA administration tool. The identity of the HA-MLM to which the component is being delegated, is part of the run-time identity of this component.

The executable code of the component is being reliably propagated to all MLMs in the HA-MLM through the group communication service (*e.g.*, Transis [2]), see Section 2.4. However, similarly to the HA-MLM controller component, this component will be executed only at one MLM at any given moment. Other MLMs within the same HA-MLM keep dormant replicas serving as warm backups for the components whose host MLM may fail, see Figure 2b. This is different from the more common practice of keeping components on a component server, and downloading them on demand. HAMSA performs component propagation as above to increase their availability.

The administrator has limited direct control over the physical location of the components within HA-MLM. This is motivated by the fact that components may need to be relocated automatically in case of failures. However, it is possible to influence the HA-MLM placement decisions by supplying some suggestive *policies* that are followed as long as no failures occur.

2.2 HAMSA-compatible Components

To render warm backups of the executing components, MLMs transparently replicate the state of the components delegated to their HA-MLM. To achieve high availability, the state is co-located with the component. This is another difference between HAMSA, and more traditional approaches in which a dedicated database is used to store the state of the components.

The state of a component consists of *Interaction State*, and *Component-Specific State*. Interaction State consists of all inbound and outbound unprocessed interactions between this component and external entities. Component-Specific State consists of arbitrary application-specific objects (*e.g.*, files) that implement a predefined interface. The state objects are managed by the components themselves. The predefined interface allows components to demand state replication without knowing any details of the replication mechanism. Similarly, MLMs handle a component's state with no knowledge of its semantics.

Components may interact with other components executing within the same HA-MLM, in different HA-MLM, and outside HAMSA, *e.g.*, with the front-ends residing in the first tier, and the network elements.

To provide its guarantees, HAMSA requires that all non-idempotent interactions between the HAMSA-compatible components, and any external entities are made through HAMSA Message Service. This allows to replicate the interaction part of a component state transparently.

However, using HAMSA Message Service is feasible only for interactions between the entities of the second and first tiers. It is not feasible to restrict in this way the interactions with the managed devices. Therefore, HAMSA is primarily targeted to monitoring, and not to other kinds of management activities that may change the state of the target devices.

2.3 HAMSA Messaging Service

Allowing a direct interactions of the components with their environment is not always safe. HAMSA defines that each component is assigned an *interaction approver* that policies its interactions. In particular, it may defer interactions with the outside entities depending on the specific state of the network, *e.g.*, when the network splits.

The interactions between the HAMSA-compatible components and the network elements are not restricted in any way. These interactions are supposed to be implicitly reflected in the component-specific state objects, which get replicated on demand from the components.

Each component is given a unique symbolic name consisting of its host HA-MLM name and a unique prefix within this HA-MLM. Each time a component is (re-)activated at an MLM, it updates the external Naming and Directory Service to renew the binding between the component's name and its communication handles. The HA-MLM assigns two types of communication handles for each component: *Mailbox*, and *Proxy*.

Mailbox is needed for directly sending a message to a component. There is one mailbox per HA-MLM that is shared by all components within this HA-MLM, and their clients. In order to support remote method invocations while using the HAMSA consistency and ordering mechanisms transparently, we use the standard proxy approach. Any remote invocation between a HAMSA component and any other party is intercepted, and processed by the per-component proxy. The proxy creates a message from the method call performed on it, and relays it to the HAMSA Messaging Service.

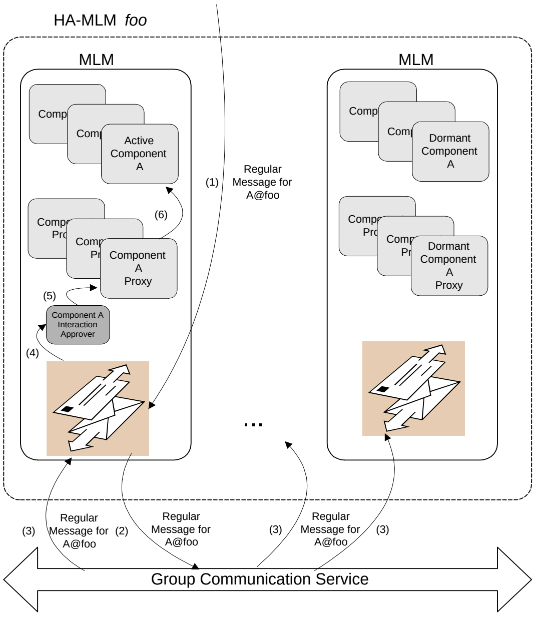

HA-MLM *foo*
MLM
Comp
Comp
Active Component A
(1)
Regular Message for A@foo
(6)
Compo Pro
Comp Pr
Component A Proxy
(5)
Component A Interaction Approver
(4)
...
MLM
Comp
Comp
Dormant Component A
Compo Pro
Comp Pr
Dormant Component A Proxy
(3)
Regular Message for A@foo
(2)
Regular Message for A@foo
(3)
Regular Message for A@foo
(3)
Group Communication Service

Figure 3. Interacting through HAMSA Messaging Service

Figure 3 depicts how a message is communicated to a component. The message is placed into the mailbox of this component by the MLM hosting it, (1). This message is not delivered to the target component immediately. Instead, (2), it is being propagated to all MLMs in the HA-MLM using the group communication service (see the next subsection). When the message is received at the group communication service level at all operational MLMs including the sender, (3), this message is assessed for delivery to the target by consulting the component's interaction approver, (4). If the interaction approver permits the interaction, the message is delivered to the active component proxy, (5). Finally, (6), the proxy delivers the message to the target component.

One restriction of this approach is that HAMSA-compatible component cannot support synchronous method invocations with non-void return values. HAMSA communication model requires that if a caller wants to receive information from a component it has to supply either a callback interface or to be registered as a recipient at the mailbox serving this component.

We use the Java programming language to implement HAMSA. We support Java RMI as an instance of the RMI technology. In our prototype, we implemented HAMSA Messaging Service as part of Java Messaging Service (JMS).

2.4 Group Communication Service (GCS)

The replication of the components state within a HA-MLM is facilitated by a group communication toolkit that is not visible outside HA-MLM (see Figure 2a). Such toolkit systems usually allow processes to form *groups* that can be addressed by a single logical name, so that messages can be sent to the group using this name as an address, and all operational members of the group receive them. HA-MLMs are realized in HAMSA as process groups.

- Reliable multicast FIFO delivery of messages.
- Per-group notification of membership changes either due to network failures, or members (*i.e.*, MLMs in the context of HAMSA) voluntarily joining/leaving the group.
- *Virtual Synchrony* model of message delivery, which, simply stated, means that members of the group that go together through the same set of membership changes receive the same set of messages.
- *Partitionable Membership Model* which means that although members of the same group can find themselves in different network partitions (due to asynchronous network splits), each connected component can continue its operation, and when a network merge occurs, the members can resume operation from a consistent point in the message stream so that the Virtual Synchrony model is preserved.

There are many group communication toolkits that supply this functionality [1].

3. Monitoring Applications

In this section we present two typical NM applications, demonstrate how one can deploy them using HAMSA, and explain the benefits NM applications gain from taking the HAMSA approach.

The first NM application is a highly available post-mortem failure analysis system. In this application, several MIB scalar variables from each network element are being kept in a centralized repository, and when a network failure occurs, the management system searches this repository for the relevant variables whose values may suggest the source for the failure (see for example [9]).

In a typical two-tier scenario such a system is deployed at a single station, and the MIB variables of all network elements are accessed from it. The collected data is kept in the local file system. When a failure of a monitored element (or of several elements from the same network region) is detected the collected data is searched and the behavior of the relevant MIB variables is examined in order to identify the cause of the problem.

In HAMSA, the centralized polling application and its repository are being handled transparently by the middleware. The administrator chooses a set of MLMs by either selecting an existing HA-MLM, or defining a new one, and delegates the polling component to this HA-MLM. Based on the component placement policy, the controller activates this polling component at one of the MLMs, while the replicas are kept for warm backup at other MLMs.

If the network splits, the monitoring continues automatically in each network partition where at least one MLM of the split HA-MLM is present. The state (*e.g.*, the collected MIB variables), is kept locally per replica of the monitoring component in each network partition. When the network re-merges these autonomously collected states become available to the administrator.

One question raised by this example concerns different configuration trade-offs available for the monitoring application that uses HAMSA. Consider the typical network configuration illustrated in Figure 4a. In this scenario, the information arrives at the monitoring station from k LANs. If the monitoring is done by centralized polling from the management station, and the connectivity to one of the LANs is lost, the monitoring of its elements cannot continue. In particular, if the failure is caused by a misconfigured access interface in the LAN's access router, the information about the cause of the problem will not be available. This is because the connectivity may be lost before the values of the router's MIB variables suggesting the cause of the problem are retrieved.

If however the administrator configures HA-MLM in such a way that there is at least one MLM per LAN, the MLM in the disconnected LAN will re-start a separate copy of the monitoring as soon as it detects (through the underlying group communication service) that there is a network partition, and all variables in the router's MIBs will be polled.

Once connectivity is re-established (say, through rolling back the configuration) the management station will be able to access this information, and the manager will be able to identify the source of the problem, (*i.e.*, a wrong configuration) and to fix it.

This example also demonstrates the importance of proper HA-MLM configuration. The administrator may be tempted to have at least one MLM in each LAN, as in our example. However, since the state of each monitoring component is distributed by HAMSA to all members of the HA-MLM, the communication costs induced by the replication may become too high.

In fact, one may choose to create k separate applications, each having a different HA-MLM containing only a pair of MLMs, as described in Figure 4b. In this case, the monitoring application for each LAN is running separately on the local MLM (according to the distance-based component placement policy), and thus being unaffected by a possible network partition. If, however, the local MLM itself fails, a copy of the

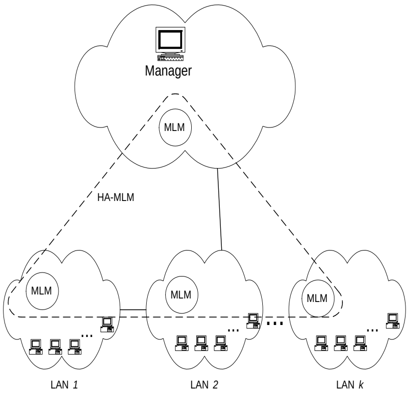


Figure 4a. All MLMs are put into a single HA-MLM

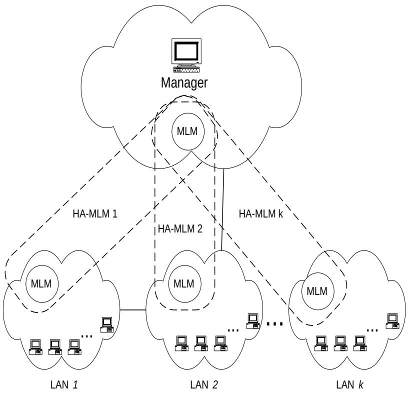


Figure 4b. Pair-wise Organization

monitoring process for that LAN will be initiated automatically by HAMSA on the MLM that is co-located with the management station. This configuration also reduces the overall monitoring traffic when there are no failures, since in this case the monitoring is done locally and the state of the monitoring components is synchronized among the two MLMs only upon the external interactions.

There exists a trade-off between the monitoring overhead traffic, and the overhead traffic induced by HAMSA due to replication it performs behind the scene. The actual amount of overhead depends on the total number of MLMs in a HA-MLM, the size of the application state in HAMSA, the frequency of external interactions, and the amount of data involved in these interactions.

For example, in the described post-mortem failure analysis application, one can choose to have a small state (*i.e.*, the serial number of the last poll), or a very large state (*i.e.*, the actual data of the last 10 minutes polling). Clearly, the latter choice allows a faster recovery after a failure of a monitoring component, but it generates much more overhead traffic. We study these trade-offs in Section 4, and show that the overhead required by HAMSA to provide the extra functionality is much smaller than the monitoring costs we saved.

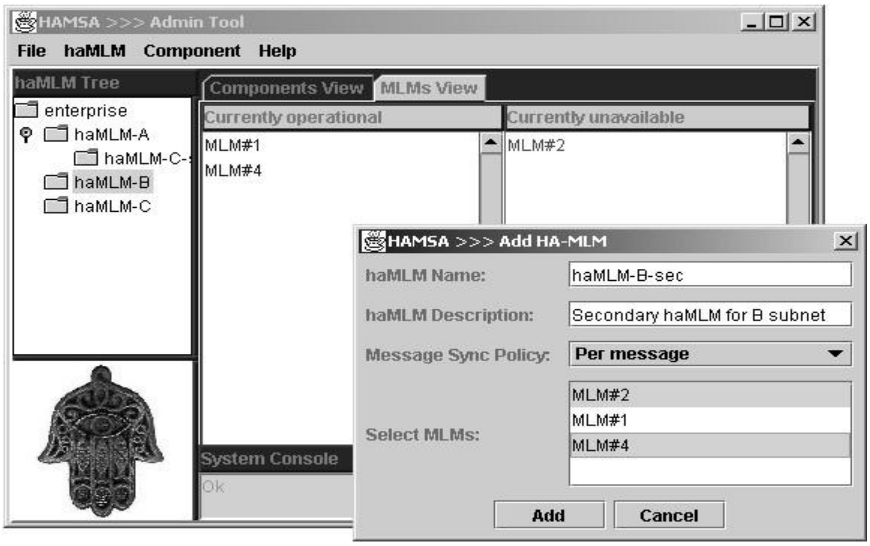
A more complex monitoring application demonstrating the inter-process communication capabilities of HAMSA is an event-driven reactive monitoring NM application. In such an application we are required to detect when a function (typically the sum) of a number of MIB variables, each belonging to a different network element, exceeds a predefined threshold (see [7]).

A centralized realization of this application involves a polling station that monitors all variables at all network elements, computes the function and sets up an alarm if the value has exceeded the threshold. This solution induces both significant traffic overhead and computation load at the monitoring station that grow linearly with the number of polled elements.

To address these issues, several algorithms that combine local computation, traps, and a centralized monitoring station were proposed in [7]. However, in order to deploy these algorithms the agent should be able to carry on simple computation tasks and issue traps, which are in many cases beyond the ability of the standard SNMP trap framework. This is a very good example where the extended functionality of HAMSA can be utilized. The global reactive monitoring application is executed in HAMSA in a distributed way. Namely, a number of copies of the same monitoring component are launched at several HA-MLMs. Each HA-MLM is responsible for its own local set of devices. According to the algorithm of [7], if a local threshold event has been detected (in this case the “local" means “with respect to the local set of variables"), then the other copies of the monitoring component are being notified using HAMSA messaging service. Then according to the algorithm, a global poll may be initiated, and, if needed, an alarm is declared. If one of the local monitoring processes fails, HA-MLM controller restarts it on another MLM, and the system continue functioning. This, of course, comes with a cost of increasing the monitoring traffic, but paying such a cost is definitely better than losing the ability to carry on with the critical monitoring task.

3.1 HA-MLM Administration

HA-MLMs are created through the HAMSA Administration Tool. A screenshot of HA-MLM creation operation is presented in Figure 5.



haMLM Name: haMLM-B-sec

haMLM Description: Secondary haMLM for B subnet

Message Sync Policy: Per message

Select MLMs: MLM#2, MLM#1, MLM#4

Figure 5. Creating new HA-MLM

The administrator picks the MLMs she wishes and groups them into HA-MLMs with a unique name. The same MLM may be a member of different HA-MLMs. A logical hierarchy of HA-MLMs may be formed. The MLMs chosen by the administrator form the *nominal view* of the HA-MLM, as opposed to the *current view*, which is always less or equal to the nominal view due to failures. MLMs that are bundled into a HA-MLM join the process group with the same name. The joining is triggered by the HA-MLM controller that multicasts a JOIN message into the ENTERPRIZE process group specifying the nominal view.

Each time a new member joins, it requests the *state* of the group from someone that is already in the group. The state versions are identified using the pair: $< current\ view >$:$< epoch\ number >$, where the epoch number is advanced each time the membership changes.

The communications cost involved in creating a new HA-MLM are dominated by the following. A single multicast through Transis is needed to propagate a *join* message. There are at most k membership change notifications delivered by Transis, where k is the size of the nominal view of the new HA-MLM. Finally, there are at most k state exchange messages needed to accommodate each newly joining MLM.

When a HA-MLM already exists, HAMSA compatible components can be delegated to it using the administration tool. The HAMSA components are realized as JAR packages.

The administrator specifies the target HA-MLM, component name, and the component specific parameters, such as the interaction approving policy, and placement policy. Two interaction approval policies are supported currently. One policy is always to approve all interactions. The second policy is to approve external interactions only if the majority of MLMs in HA-MLM are present in the network partition.

Placement policy defines either load-dependent, or distance-dependent placement of a component. These policies are best-effort ones.

The communication cost involved in the component delegation protocol is dominated by multicasting a component through Transis to all MLMs of the HA-MLM, and obviously, depends on the size of the component.

4. Performance Evaluation

In order to understand the trade-off between the communication overhead induced by HAMSA, and the possible reduction in monitoring overhead, consider again the scenario described in figures 4a, and 4b. We want to compare the amount of traffic overhead generated by the monitoring application without HAMSA with the overhead induced by HAMSA and the underlying group communication service.

The group communication service is responsible for failure detection that is based on periodic broadcasting of short *I-am-alive* messages. In general, this overhead grows as k^2, where k being the size of HA-MLM. Optimizations that reduce by factor l, the number of LANs, are possible [3]. However, this is inevitable overhead of failure detection that cannot be strictly attributed to HAMSA or group communication, because any application wishing to achieve the high availability guarantees of HAMSA on its own would pay these costs anyway. The experiments performed in [3] with the current implementation of Xpand and Transis show that group communication scales to 200 hosts dispersed over WAN without visible impact on the regular traffic.

As described in Section 3 the overhead of HAMSA itself strongly depends on the way we configure HA-MLMs and on the size of the application *state*. In order to investigate the trade-off, assume that the state size sent by HAMSA is 150 bytes. This is a reasonable size, when one chooses to use a small state (like a measurement sequence number).

Figure 6a depicts the tradeoff for the two choices of HA-MLM configuration and for 10, and 20 scalar MIBs variables in each LAN. We assumed here that due to the SNMP encoding (SMIv1/SMIv2), polling of one variable takes about 150 bytes, and thus polling 10 or 20 variables per LAN will consume 1500, and 3000 bytes respectively for each LAN. HAMSA's overhead depends on the HA-MLM configuration. If all k MLMs are members of the same HA-MLM, we need to update all k states each polling interval. This takes $150 * k^2$ bytes. On the other hand, if we use k different HA-MLMs, each of size 2, HAMSA's overhead is reduced to $150 * k$.

One can easily see that even for a very small number of monitored variables the overhead of HAMSA is significantly smaller than the monitoring overhead of a traditional application. This is a big advantage even without considering the HAMSA's main goals: extended functionality and reliability.

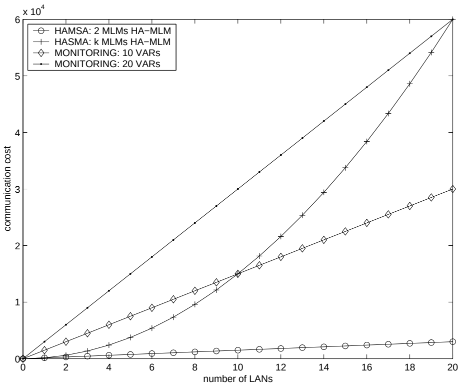


Figure 6a. HAMSA communication cost, and monitoring communication cost as a function of the number of LANs.

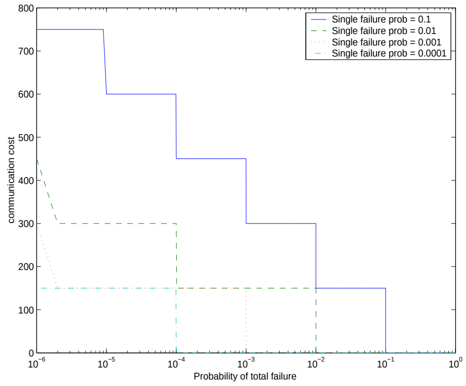


Figure 6b. HAMSA communication cost per state change as a function of the required system error failure probability.

The main mission of HAMSA is increasing the system reliability. However, the high reliability comes with the cost of introducing more MLMs. In particular, this implies a higher communication overhead. Thus, there is a trade-off between the level of availability and the traffic overhead. In order to evaluate this trade-off, we consider the same scenario as above.

The current host MLM of an active component replica propagates its state to the rest of its HA-MLM through multicast. Thus, communication cost of HA-MLM replicating the state is linear in the number of group members. However, when we increase the number of MLMs in the group, we reduce the probability of a total system failure, since HA-MLM restarts the failed process on a different MLM as long as they are available.

Thus, if we have n MLMs in a HA-MLM, and the independent probability of a single MLM failure is p, the probability of the application failure is $\tilde{p} = 1 - p^n$. Since we have only one active component per network partition then the communication is $s * (n - 1)$ per each state in the component state, where s being the component state size. To obtain specific numbers, let $s = 150$ bytes, as in our example. Then, in order to get an application failure probability of $\tilde{p}$ we pay $150(\lceil \frac{\log(1-\tilde{p})}{\log p} \rceil - 1)$ bytes per change. This cost is plotted in Figure 6b, for single MLM failure probability of 0.1, 0.01, 0.001, and 0.0001.

As one can see, in order to get the often desired "5 nines" reliability, starting from a very high error rate of 0.1 on a single machine, 6 MLMs are sufficient. The communication cost ($150(6-1) = 750$ bytes per state change) becomes much smaller when the reliability of a single machine increases.

5. Background and Related Work

The quest for more efficient and versatile management paradigms has been pursued by many researches over the last few years. One general line of approach suggests using mobile agents, active networks, or programmable networks for decentralizing and shortening the control loop [4, 13]. Usually, these proposals focus on the mechanics of the mobility and extended functionality rather than on the high availability and meta-management issues being in the focus of this paper.

Several approaches for integrating the management by delegation approach [17] into SNMP environment have been proposed recently [10]. With the advent of Java, the delegation is easily implemented by exploiting its mobility and security features making Java a preferred language for developing delegated programs.

Java Management Extension (JMX) [11] is an emerging Java standard for representing managed objects as Java Beans. JMX Bean is an object that serves as a Java wrapper facade for the actual managed object. JMX Beans may be co-located with the objects they represent at the agent side, or be deployed in a distributed fashion. In the latter case, JMX Beans need distributed object services of the second tier that are currently left unspecified by JMX. HAMSA components can be implemented as JMX Beans.

One of the more mature Java technologies for deploying three-tier Java applications is provided by Enterprise JavaBeans (EJB) [8]. EJB defines interfaces for Application Server, and Enterprise Java components (Beans) that execute in the environment of the application server that manages transactions, persistency, security, and naming services for the components.

The problems that HAMSA copes with are very similar to those of the stateful EJB clustering. Some of the existing EJB implementations provide fail-over models that allow replication of the beans' states, and support takeover of the failed beans by other servers in the cluster [8]. Most EJB servers perform stateful fail-over by using either in-memory replication, or persistent storage to a shared database. These solutions are inappropriate for the NM domain, since they rely on the fact that the network remains connected. To the best of our knowledge, there is no current implementation of EJB, or other application server technology that provide the high availability of the second-tier components execution to the level that allows their comparison with HAMSA.

6. Conclusion and Future Work

Efficient monitoring of large and dynamic distributed systems becomes challenging. Current standard technologies scale poorly due to their inherently centralized approach. We present a lightweight monitoring middleware called HAMSA that dynamically allows to enhance monitoring functionality, and decentralize it in a reliable and efficient manner. This work presents the architectural overview of the middleware, and the possible functional and performance trade-offs involved in its deployment. Our architecture uses a group communication middleware to increase availability, modularity, and scalability.

We are currently testing our implementation under different load conditions, and for different failure scenarios. The exetensive performance evaluation study will be presented in the full version of the paper.

7. Acknowledgments

We thank Elias Procopio Duarte, Jr, and Aldri L. dos Santos for their valuable comments.

References

[1] ACM. *Communications of the ACM, special issue on Group Communication Systems*, 39(4), April 1996.

[2] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication sub-system for high availability. In *22nd Annual International Symposium on Fault-Tolerant Computing*, july 1992.

[3] T. Anker, G. Chockler, D. Dolev, and I. Shnaiderman. The design of xpand: A group communication system for wide area. Technical Report HUJI-CSE-LTR-2000-31, The Hebrew University, July 2000.

[4] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile agents for network management. *IEEE Communications Surveys*, 1(1):2–9, Forth Quarter 1998.

[5] D. Breitgand, G. Shaviner, and D. Dolev. Towards highly available three-tier monitoring applications (extended abstract). `http://cs.huji.ac.il/~davb/abstracts/hamsa.ps`, 2000. 11th IFIP/IEE Internationaal Workshop on Distributed Systems: Operations and Management, Austin TX, USA.

[6] M. Daniele, B. Wijnen, M. Ellison, and D. Francisco. Agent extensibility (AgentX) protocol, January 2000. RFC 2741.

[7] Mark Dilman and Danny Raz. Efficient reactive monitoring. *IEEE Journal on Selected Areas in Communications (JSAC), special issue on recent advances in network management*, 20(4):668–677, May 2002.

[8] Roman E., Ambler S., and Jewell T. *Mastering Enterprise JavaBeans(tm)*. Wiley, 2nd edition, 2002.

[9] E. P. Duarte Jr. and Aldri L. dos Santos. Semi-active replication of snmp objects in agent groups applied for fault management. In *7th IEEE/IFIP International Symposium on Integrated Network Management IM*, Seattle, WA, May 2001.

[10] D. Levi and J. Shonwalder. Definitions of Managed Objects for the Delegation of Management Scripts, May 1999. RFC 2592.

[11] Sun Microsystems. Java management extensions(JMX) instrumentation and agent specification, v1.1. `http://java.sun.com/products/JavaManagement/doc.html`, mar 2002.

[12] B. Pagurek, Y. Wang, and T. White. Integration of mobile agents with SNMP: Why and how. In *2000 IEEE/IFIP Network Operations and Management Symposium*, pages 609 – 622, Honolulu, Hawaii, USA, April 2000.

[13] Danny Raz and Yuval Shavitt. Active networks for efficient distributed network management. *IEEE Communications Magazine*, 38(3), March 2000.

[14] Marcelo Gonçalves Rubinstein and Otto Carlos Muniz Bandeira Duarte. Evaluating tradeoffs of mobile agents in network management. *Networking and Information Systems Journal*, 2(2):237–252, 1999. HERMES Science Publications.

[15] William Stallings. *SNMP, SNMPV2, SNMPV3, and RMON 1 and 2*. Addison-Wesley, January 1999.

[16] Y. Yemini. The OSI Network Managemnt Model. *IEEE Communications Magazine*, pages 20–29, may 1993.

[17] Yechiam Yemini, German Goldszmidt, and Shaula Yemini. Network Management by Delegation. In *The Second International Symposium on Integrated Network Management*, pages 95–107, Washington, DC, USA, April 1991.