Providing Automated Detection of Problems in Virtualized Servers using Monitor framework

Gunjan Khanna, Saurabh Bagchi
School of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN, USA
(gkhanna, sbagchi)@purdue.edu

Kirk Beaty, Andrzej Kochut, Gautam Kar
IBM T.J. Watson Research Center,
Hawthorne, NY, USA
(kirkbeaty, akochut, gkar)@us.ibm.com

***Abstract* —Increasing management costs with large servers have caused adoption of a technique called *Server Virtualization* whereby multiple virtual machines can be hosted on the same physical machine. Reduction in the number of physical machines and amount of rack space brings the total cost of ownership down. Virtual servers residing on the same physical machine are likely to suffer resource contention under varying workload conditions. We are targeting distributed environments where some services are executed on virtual servers. Existing management frameworks fail to address the problems faced in such environments because of a narrow focus only at system metrics which are not sufficient to explain the application problems. Also the management frameworks cannot detect problems arising out of virtualization. In this paper we propose a hierarchical, scalable, non-intrusive Monitor framework which addresses not only system level problems but also problems in application semantics. We show how design of Monitor framework is different from existing management frameworks making it suitable for virtualized environments. We demonstrate the specific class of problems which the Monitor can detect in a virtualized environment running an e-commerce application.**

***Index Terms:* Server virtualization, Application performance management, Error detection, Monitor framework.**

I. INTRODUCTION

Traditionally business units used to deploy a new machine for every new application. This has caused an unplanned growth of servers most of which are underutilized. This server sprawl has caused the management cost to increase manifold. The costs arise due to space and human skills required for these complex systems. V*irtualization* is a new architectural approach which is getting widely used in several business centers. Providing an abstract layer between the operating system and the hardware so as to isolate the two is referred to as server virtualization. It allows users to install multiple operating systems on a single physical machine with perfect isolation from one another. A virtual system has its own set of virtualized hardware components (like CPU, NIC etc.). The layer between the virtual machine and the physical machine that arbitrates access to resources is called the *hypervisor*. In this paper, we specifically address the problem of management of virtualized servers which are created through *server consolidation* (see Figure 1). Since these virtualized servers are running applications together on the same physical hardware, they are more susceptible to complex performance problems as compared to stand-alone servers.

Management issues in a virtualized server environment encompass initial configuration, resource allocation, load management, and problem debugging. Since management systems and virtualization have evolved in different evolution paths, this leads to a wide disparity between the current management frameworks and the virtualization layer. For example, traditionally the management layer had complete control over the operating applications but with virtualization, the underlying hypervisor has several control tasks which the management layer is unaware of. In [11] the authors discuss the wide disparity which exists between the current virtualization technologies and existing management frameworks thus hinting at developing new management infrastructures to handle the virtualization environments.

Arguably, a possible solution for system management in virtualized environments is to use existing management agents on these virtual servers to obtain the virtual system's metrics. Management agents (like IBM Director Agents[10]) are software pieces which are co-located with the system which needs to be monitored. They periodically (or on poll) collect system metrics and send them to a central server. Unfortunately, virtualization hides the actual CPU cycles and memory utilized on a host if the management agent is installed on a virtual machine. In order to be able to obtain some reasonable estimate of these metrics, management agent(s) must be installed on the physical machine as well. Hence in order to be able to make any conclusion about the performance problems of an application running on a virtual machine, data collected from the two agents on the physical and the virtual systems must be correlated.

Difficulty in detecting problems in resource allocation can get exacerbated if the hypervisor itself employs dynamic resource re-adjustment mechanisms. In a distributed system, there are complex interactions between the applications which can cause error propagation making diagnosis harder. For example in Figure 1, assume App1 and 2 are running WebSphere while App3, 4, and 5 are running replicated DB2 service to provide the database for applications running on WebSphere. A problem in App2's load balancer can cause it to direct all the DB2 connections to App3, leading to a high resource allocation of App3. Traditional management agents sitting at App2 or App3 will not be able to detect this *error propagation* from App2 to App3. Customer load on these applications also keeps varying and without any knowledge of workload profile it is impossible to know the distribution of resource demands of different applications *a priori*. Due to simultaneous peaks in the workloads of two applications (say App*m* and

App*n*) residing on the same physical machine, transient resource contention could occur, potentially causing both applications to violate Service Level Agreements(s) (SLAs).

Problems in virtualized servers can be classified into various categories for example: performance problems vs misconfigurations, application level vs system level problems, and syntactic vs semantic problems. Application level problem could be a simple deadlock in DB2 table, while a system level problem could be high CPU utilization of a server. Typically semantic problems are hard to detect because of the complex system interactions. This gets further exacerbated in virtual scenarios because of the sharing of resources. Current management systems only concentrate on system parameters but as is evident from the example of the load balancer, detection and diagnosis of such problems is not feasible by simply examining the system metrics. There is a need for mapping between the application behavior and system metrics. These mappings not only depend on the application but also on the architecture of the underlying hardware. Thus determining application problems through simply looking at system metrics is not accurate and in most cases not feasible. New management architecture is needed to judge performance problems by taking into account both the application level semantics and the system level semantics.

We propose such a management framework in the form of hierarchical Monitor architecture which constantly monitors and *verifies* the behavior of the applications running on a virtualized server environment against a rule base. This framework has been demonstrated by us for error detection in [12]. The framework consists of Local, Intermediate and Global Monitor(s) organized in a hierarchy with a single Global Monitor (GM) at the root. Each Local Monitor (LM) is responsible for monitoring the local interaction of a domain which specifically would be all the applications running in virtual servers on a single physical machine. The monitoring takes the form of matching the interactions against a rulebase of anomalous rules. Intermediate Monitor(s) (IM) is responsible for monitoring interactions between applications residing on different virtual machines. Higher level Monitor(s) is responsible for correlating and performing verification over the aggregated information to ensure that inter-physical machine interactions are correct. Monitor needs to obtain the messages which are exchanged between the applications to perform verification. A Local Monitor is either placed on the hypervisor which has virtual servers running on it or it can be placed in the vicinity with only a simple packet forwarding functionality sitting at the hypervisor to obtain these messages. Figure 4 illustrates the hierarchical architecture of the Monitor. The Monitor performs verification of application entities (e.g., applications running on virtual servers in the context of this paper) against a rule base which is provided as an input. In order to obtain the system level metrics like CPU, memory etc., a management agent is installed on the virtual server which reports these metrics to the Local Monitor sitting on the hypervisor. The Monitor uses both system level metrics obtained from the management agents and application level semantics through the observed messages to perform verification. The Monitor's rule expression infrastructure is rich and generic enough to address most of the common problems faced in virtual server scenarios. Detection of an error triggers the *diagnosis* process which aims at pin-pointing the application which is the root cause of the problem. For the scope of this paper we restrict ourselves to problem detection in virtualized server scenarios.

In this paper we make the following contributions:

- Applying a non-intrusive scalable Monitor framework for problem detection in virtualized server scenarios.
- Architecture for correlating the information from various local segments thus providing a global detection framework.
- Providing stateful detection using both system and application level semantics.
- Demonstrating how detection happens for a class of common problems in IBM's virtualized environment.

II. SYSTEM MODEL

A. Virtualized Server(s)

Figure 2 shows an example virtualized scenario on a blade center. Blades in the IBM BladeCenter are the physical machines which host the virtual servers. In the example each of the three blades has a hypervisor (for example VMWare ESX) installed. For each blade, the Virtual Machines (VM_i) are created on top of the ESX server giving each of them equal shares of the physical CPU by default. A hypervisor can be configured to give preferential shares to a particular virtual machine, though in today's commercial systems, the allocation cannot be varied at runtime. The BladeCenter is connected to a storage area network (SAN) which provides shared storage for all the blades. The virtual machine images are stored in the SAN. Shared storage is only necessary to provide seamless mobility of OS images from one physical blade to another physical blade using VMotion, a product offering from VMWare that lets users move live, running virtual machines from one host to another while maintaining continuous service availability. An example management framework for such an environment could be IBM Director, consisting of agents and a management server. IBM Director Agents are installed on each hypervisor and on the VMs as well. The IBM Director Server sits on a separate machine and pulls management data from the director agents. The management data consists of metrics like CPU utilization, memory, I/O, etc. In a typical deployment, each virtual machine runs only a single instance of an application. The two primary applications are IBM's Websphere Application Server (WAS) and DB2 database server.

Before we propose the architecture for management of virtualized server(s), we provide some detail on the generic Monitor architecture and explain how it is different from existing management frameworks.

B. Monitor Architecture

The Monitor(s) snoops over the communication between the interacting applications and performs matching of the observed communication against a rule base that characterizes *acceptable* behavior.

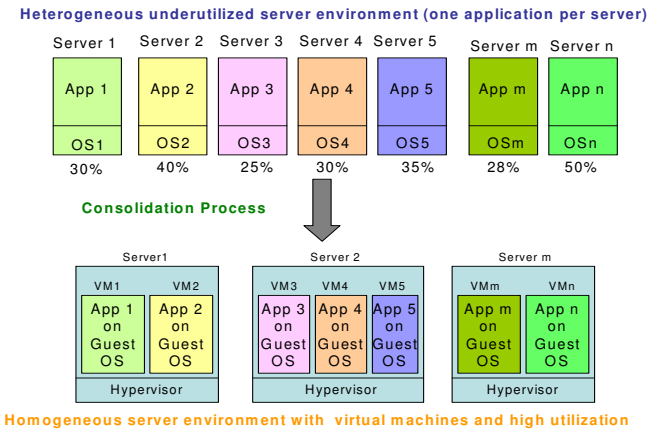


Figure 1: A Typical Server Virtualization Scenario used for Server Consolidation

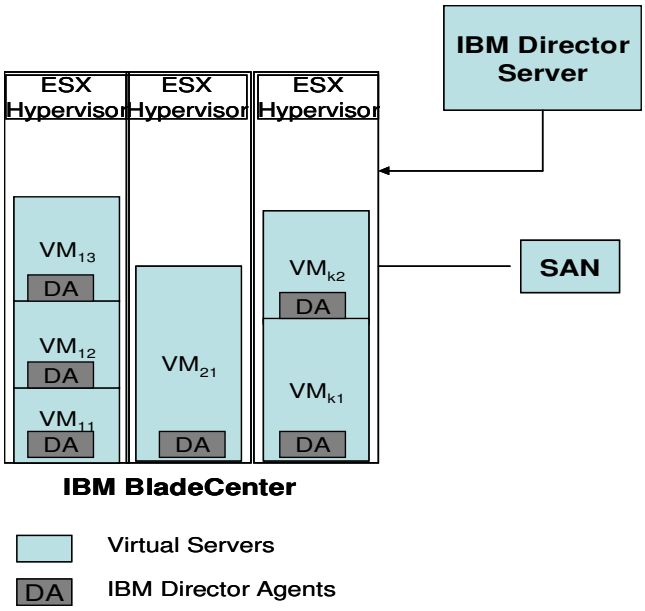


Figure 2: IBM's virtualized server scenario on a BladeCenter

The individual servers can be treated as black boxes and their internal state transitions are invisible to the Monitor. Monitor has several modules including Data Capturer, State Maintainer and Detection Engine (see Figure 3). Data Capturer is responsible for obtaining the messages exchanged by the interacting applications. State Maintainer maintains a Monitor view of the application state through information extracted from the observed messages. Detection engine is responsible for detecting problems in the application. More details on Monitor's individual components can be found in [13]. The Monitor architecture is generic and applicable to a large class of message passing based distributed applications which might be running on the virtual server. It is the specification of the rulebase that makes the Monitor specific for that application. We provide a specification syntax for the rules, in which a rule may be combinatorial (valid for all points in time in the lifetime of the application) or temporal (which have an associated time component). We provide fast rule matching algorithms that match the incoming messages against the rules. In order to make the infrastructure scalable, efficient, and accurate, we develop a hierarchical Monitor structure where the Local Monitors directly gather communication between the interacting applications that are geographically localized and the higher level Monitors get processed and filtered messages from the lower level Monitors (Figure 4). This allows a higher level Monitor to perform detection using observed behavior that may not be local. Monitors are replicated at each hierarchical level to obtain fault tolerance from transient faults within the Monitor(s).

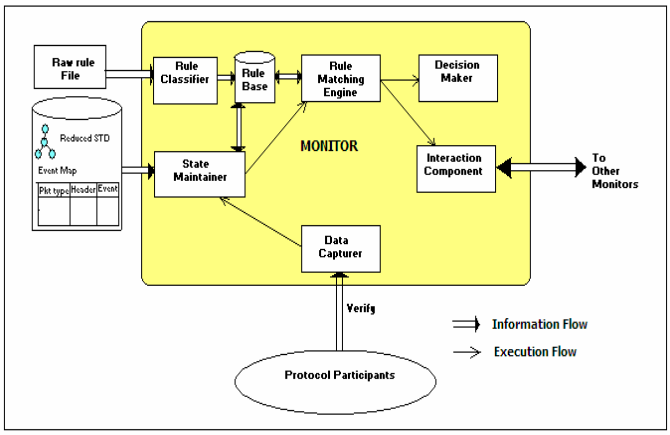
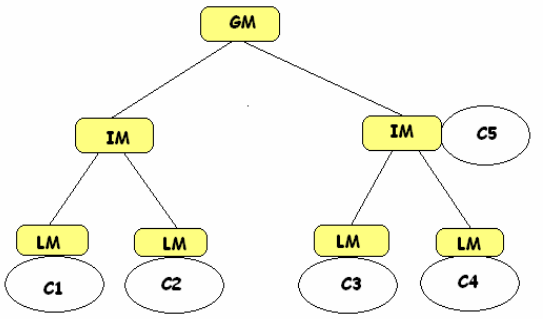


Figure 3: Monitor architecture with process flow and information flow among multiple components



C: Clusters consisting of each Hypervisor; LM: Local Monitor; IM: Intermediate Monitor; GM: Global Monitor

Figure 4: Example topology of local, intermediate, and global monitors

C. Differences from the existing Management Frameworks

There are several management frameworks available to provide management tools to a system administrator, such as IBM Tivoli [6] and HP OpenView [16]. Broadly most of the management frameworks consist of two parts: an Agent and a Server. The agent is co-located with the machine which is running the application and mostly the agent is custom-built for each kind of application. The Agent is responsible for sending system metrics like CPU, memory and disk usage to the server which is placed on a separate machine in the vicinity. Currently the management frameworks are centralized with the server collecting all the information and using heuristics to detect bottleneck events. The Monitor architecture differs from the existing Management frameworks in several aspects, the most prominent ones are as follows:

- Existing frameworks require co-located agents which measure the system metrics to deduce performance problems. For e.g., a Director agent would report performance problem on measuring a high CPU utilization. While the Monitor(s) perform verification of the application level semantics along with system level metrics, ensuring that application is functioning properly. For example: if there is a problem in an EJB and it causes a DB2 table to get locked then a management module which is only looking at the system metrics will not be able to diagnose it. Monitor needs to correlate the information between the application and system layer.. As explained in the architecture in section II. B., the Monitor *verifies* the messages exchanged by the application entities against a rulebase to detect errors. Due to the above difference the Monitor perceives the application as a black-box and performs both detection and diagnosis in a non-intrusive fashion. This requires no change to the application for management purposes.
- In contrast to the central architecture, Monitor has a hierarchical architecture which performs filtering of local messages at the lower levels. Each local Monitor verifies the messages which are required for local rules and only sends aggregate information for the other messages. This aggregate information which is collected by the higher level Monitor is used for detecting violations in the application across the local domains. For example: If there are multiple clusters running a common application, a system administrator might restrict the number of customers to say 10 and overall the total customers can be allowed to 100. In such a scenario each local Monitor will verify for the cluster within its domain the number of customers. While an aggregate count of the customers would be sent up from each local Monitor to a higher level Monitor (say IM or GM) who would verify that the total customer count is below 100. This provides scalability with the increasing number of application entities. Since each Monitor only checks for some rules, the load on the Monitor is restricted. A hierarchical framework is also important because the application can have different behavior in the local domain compared to the global perspective. Since in a common virtualized environment, most of the interactions will be local, most of the interactions are only matched at the LM and do not reach the higher level Monitors.
- Current management frameworks are quite rigid and once set up, require intensive human interface to adjust to changes. The Monitor(s) on the other hand are re-configurable to new application or environment, providing autonomic functionalities and are fault tolerant. Only the appropriate rulebase needs to be loaded into the Monitor.

III. MONITOR(S) FOR VIRTUALIZED SERVER ENVIRONMENTS

A. *Challenges*

Management of servers in a virtualized environment poses several challenges to the existing management frameworks. Some of the common challenges faced in detection of performance problems are as follows:

1. How to quickly detect the problem that arises out of the resource contention between multiple applications running on the same host?
2. How to find out error propagation real-time during the execution of the applications?
3. In virtualized scenarios there might be local interactions between the applications residing on the same hypervisor and inter-hypervisor interactions both of which need to be addressed in a *scalable* fashion.
4. A local determination of problem within a virtual server is likely to be misleading. For example, an agent on one of the virtual servers may be reporting that CPU cycles are not available, while there may be idle cycles at the hypervisor. How to correlate the information about the system metrics with the application state information obtained through monitoring of messages?
5. How to correlate the information about the system metrics obtained from the virtual machines with the system information obtained from the hypervisor (i.e. the actual physical machine)?
6. How to detect system resource level and application level dependencies between the various virtual servers running on the same physical hardware?

B. *Proposed Solution*

We propose to use the Monitor architecture for detecting performance problems (or errors configuration). As depicted in Figure 5, each hypervisor has a local Monitor placed which would verify the local interactions between the applications running on that hypervisor. Here we need to co-locate the Monitor at the same host because when virtual servers are installed on a hypervisor, a virtual switch is created to route packets between the virtual servers and is bound to one of the NICs present on the machine. Communication through "network packets" exchanged between the servers goes through this virtual switch. Since the packets never leave the physical machine, it is difficult to snoop over these packets without active support from the applications or the hypervisor (e.g., a simple trap on the hypervisor which copies the packets on the virtual switch and forwards to the Local Monitor). It is considered least intrusive to have the Local Monitor co-located on the machine because no change is required to the hypervisor. Assuming a Local Monitor is residing on each hypervisor, we can have multiple hypervisors forming a cluster. Intermediate Monitor is placed in the vicinity preferably the same LAN for verification of messages exchanged between the applications on different hypervisors. The architecture can be extended to have several IMs and a Global Monitor too based on the size of a virtualized environment. A common rule base and state transition diagram of the applications verified is provided as an input to the Monitors at all levels.

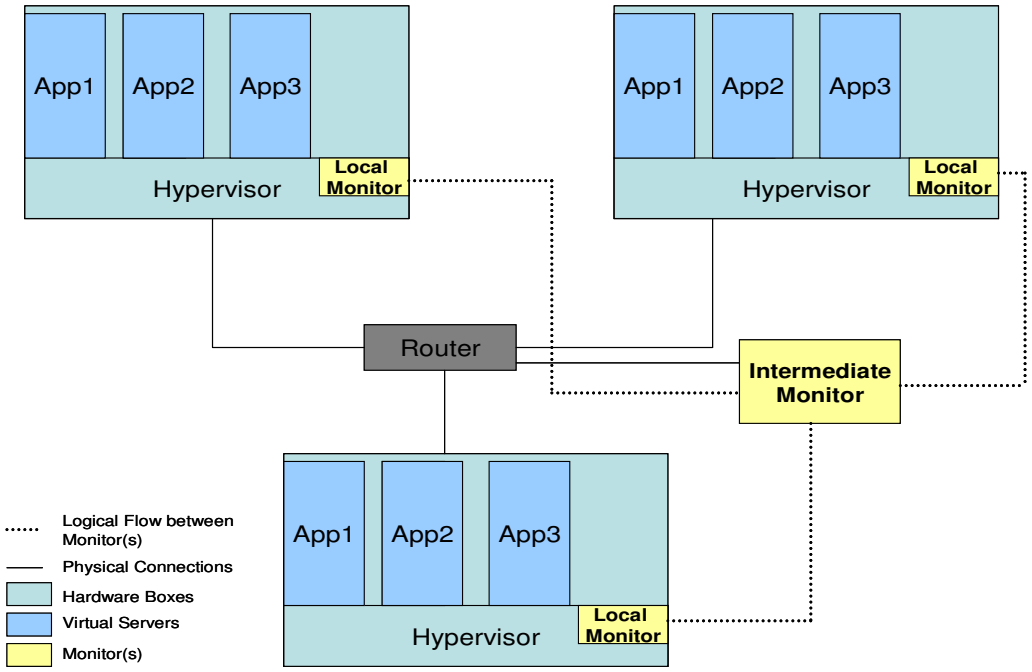


Figure 5: Illustration of Monitor usage for managing virtualized server environment

Rules can be formed of the *states, events, state variables* and *time of transitions*. Each state has a set of state variables. Events may cause transitions between states. In our context, events are message sends and receives which an application running on virtualized system may perform. The rulebase consists of Temporal and Combinatorial rules. Temporal rules are valid for some time duration while the combinatorial rules are valid for all times. Temporal rules can be divided into the following five categories that seem adequate for a large class of anticipated disruptions and protocols.

Type I:

$S_p = \text{true for } T \in (t_N, t_N + k) \Rightarrow S_q = \text{true for } T \in (t_I, t_I + b)$

The above rule represents the fact that if for some time k starting at t_N, a state S_p is true, then it will cause the state S_q to be true for some time b starting from t_I. The time t_N represents a time when some defined event E_I takes place.

Type II: S_t is the state of an object at time t : $S_t \neq S_{t+\Delta}$, if event E_i takes place at t. The state S_t will not remain constant for more than Δ time units if an event E_i takes place.

Type III: $L \leq |V_t| \leq U \;\; t \in (t_i, t_i+k)$

The state variable V_t in a particular state S_I will be bounded by L and U in some time k starting at time t_i when the defined event corresponding to the rule first occurs.

Type IV: $\forall t \in (t_i, t_i + k)\; L \leq |V_t| \leq U \Rightarrow L' \leq |B_q| \leq U' \;\; \forall q \in (t_n, t_n + b)$. A state variable V_t being bounded by upper and lower bounds in time k will cause another state variable B_q to be within some bounds and will hold true for some time interval b. This rule is in fact the master rule and the three previous rule types are special cases. But we still need the first three rule types because matching this class of rule entails matching more variables, which incurs higher latency than the first three classes.

Type V:

$s = S_i \; \forall t \in (t_0, t_0 + \alpha) \Rightarrow s \neq S_i \; \forall t \in (t_0 + \alpha, t_0 + \beta);$

$s.t. \; \beta > \alpha$

This rule prevents a state transition from S_i back to the same state. Rule format allows ease in expressibility of common problems faced in virtualized scenarios. They also help in describing system and application level semantics together.

Monitors perform automatic rule segmentation to deduce the rules relevant for its verification domain. For example:

1. *Local Rule*. A local rule is matched at the current Monitor only and does not cause any event to be generated for matching at any other Monitor.
2. *Global Rule*. A global rule generates event(s) to be forwarded for subsequent matching at other Monitors.

C. *Problems Detected using Monitor Architecture*

The Monitor can detect a large set of problems occurring in virtualized scenarios including application and system level problems. For example, consider the three applications running on a hypervisor in Figure 5. Lets assume that S_1, S_2, S_3 correspond to the states with peak resource consumption which cannot be supported on the same physical machine. From Figure 5 we can see that these applications should not achieve peak resource consumption at the same time i.e., $\neg(S_1 \wedge S_2 \wedge S_3)$ forms a combinatorial rule to detect the system level problem. An example of application level problem could be table lock contention at a common DB2 server serving multiple WASs. One could have a temporal rule which restricts the number of simultaneous open DB2 connections between $[L_0, U_0]$. Formally stated:

$\exists t = T \; s.t. \;\; S_i = true \Rightarrow \; L_0 \leq O_c \leq U_0 \;\; \forall t \, \varepsilon \, (T, T + \delta)$

; where S_i is a state of DB2, $\underline{O_c}$ are the number of open connections between time $(T, T+ \delta)$. The rule states that if the DB2 application enters a state S_i then the number of open connections must be restricted between $[L_0, U_0]$.

Because of the hierarchical structure, Monitor(s) can correlate messages (and alarms) from Local Monitor(s) to provide increased coverage. Consider in Figure 5 that one of the hypervisor has a WAS server running which

connects to 3 replicas of DB2 instances running on the second hypervisor. If there is a fault present in WAS which causes it to contact only one of the replicas, there will be higher resource utilization for that replica. A rule which forces near equal resource utilization for all the replicas would get violated at the Local Monitor. Additionally since other DB2 replicas are not seeing any messages, it would cause an appropriate rule to flag in the Local Monitor sitting on that hypervisor. Since an Intermediate Monitor would be verifying these inter-hypervisor messages, it can correlate the alarms from Local Monitors. This is also an example where the Monitor(s) correlate the information from the system metrics with the application state. Another example could be where a temporal rule consists of the number of customer requests in WAS and its CPU utilization like

$$If \;\; C_0 \le C_W \le C_1 \;\; \forall t \, \varepsilon (T, T+\delta)$$

$$\Rightarrow L_{CPU} \le WAS_{CPU} \le U_{CPU} \;\; \forall t \, \varepsilon (T, T+\ell)$$

Here the Monitor is using the system level information (CPU) with the application data (Customers) to perform problem detection.

D. Addressing Challenges

We will briefly discuss why the Monitor architecture is able to handle the challenges raised by the virtual server scenarios.

1. Local Monitor sitting on each hypervisor collects the system level metrics of the physical machine and the virtualized server(s) hosting the application(s). Monitor performs local aggregation of this information and detects resource contentions. Combinatorial rules such as $\neg(S_1 \wedge S_2 \wedge S_3)$ (explained before) can detect if there is resource contention at the applications. Further Local Monitor can also detect if a particular application needs more resources by correlating the system metrics with the monitored messages. For e.g., if a WAS application is requesting for a lot of DB2 connections, or the number of customers have increased, which is causing the resource contention (monitored as system metrics). A temporal rule

   $$\exists t = T \; s.t. \;\; S_k = true \Rightarrow C_0 \le C_W \le C_1 \;\; \forall t \, \varepsilon (T, T+\Delta)$$

   where C_w represents the number of simultaneous customer requests at WAS, for detection.
2. Due to the hierarchical architecture the Monitor(s) verifies the local interaction at the Local Monitor and filters these messages. Only aggregate information about messages which are inter-hypervisor are passed to higher level Monitor (like IM in Figure 5) for verification. Local filtering and load sharing between the Monitors, ensures the scalability of the framework. Because the Monitor(s) only views the external messages and system metrics, it requires no change to the application or the hypervisor. The Monitor can perform the rule matchings efficiently making it an online detection framework (see [13]).
3. As part of ongoing work, we are exploring diagnosis by the Monitor. Due to the capability of correlation, the Monitor can perform some diagnosis through observing the causal order of events (messages in this context). Causal trace back of these events and their verification could aid in diagnosis of the faulty application.

IV. CONCLUSIONS

In this paper we show how the hierarchical Monitor architecture can be used to solve specific application problems in virtualized server scenarios. As a part of current ongoing research we are investigating how to extend this architecture for performing diagnosis. The big question is how to identify the applications that are causing the resource contention, without putting intrusive instrumentation in the applications. We have proposed a generic diagnosis framework [15] and it will be applied to the application scenario described here.

REFERENCES

[1] C.A. Waldspurger, “Memory resource management in VMware ESX server,” Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), 2002.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” Symposium of Operating Systems Principles, 2003.

[3] http://www.vmware.com/

[4] R. J. Figueredo, P. A. Dinda, and J. A. B. Fortes, “A case for Grid Computing on Virtual Machines” Proceedings of the 23rd International Conference on Distributed Computing Systems, 2003.

[5] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum, “Cellular Disco: resource management using virtual machines on shared memory multiprocessors”, 17th ACM Symposium on Operating Systems Designs and Principles (SOSP’99), 1999.

[6] http://www-306.ibm.com/software/tivoli/.

[7] http://www.vm.ibm.com/overview/zvm52sum.html.

[8] http://www-1.ibm.com/servers/eserver/zseries/zos/bkserv.

[9] http://www-1.ibm.com/servers/eserver/iseries/scon.

[10] http://www-1.ibm.com/servers/eserver/xseries/ systems_management/director_4.html.

[11] Sven Graupner, Ralf König, Vijay Machiraju, Jim Pruyne, Akhil Sahai, and Aad van Moorsel “Impact of Virtualization on Management Systems”, Tech. Report HP Labs, HPL 2003-125.

[12] G. Khanna, P. Varadarajan, S. Bagchi, "Automated Online Monitoring of Distributed Applications Through External Monitors," accepted for publication in IEEE Transactions on Dependable and Secure Computing (TDSC), 2006.

[13] G. Khanna, P. Varadarajan, and S. Bagchi, "Self Checking Network Protocol: Monitor Based Approach,", published in Symposium on Reliable and Distributed Systems, (SRDS), pp. 18-30, , 2004.

[14] G. Khanna, K. Beaty, A. Kochut, and G. Kar, “Dynamic Application Management to address SLAs in a Virtualized Server Environment,", accepted in Network Operations and Management (NOMS), 2006.

[15] G. Khanna, P. Varadharajan, M. Cheng, and S. Bagchi, “Automated Monitor Based Diagnosis in Distributed Systems,” Purdue ECE Technical Report 05-13, August 2005.

[16] http://www.managementsoftware.hp.com/