Adaptive Monitoring: A Framework to Adapt Passive Monitoring using Probing

Deepak Jeswani
IBM Research Lab,
New Delhi, India.
Email: djeswani@in.ibm.com

Maitreya Natu
Tata Research Development and Design Centre,
Pune, India.
Email: maitreya.natu@tcs.com

R. K. Ghosh
Indian Institute of Technology,
Kanpur, India.
Email: rkg@cse.iitk.ac.in

***Abstract*—Effective use of monitoring tools demands an understanding of monitoring requirements which system administrators most often lack. In this paper, we propose to replace today's ad-hoc, manual, intuition-based approach with a more systematic, automated, and analytics-based approach for system monitoring. We propose an adaptive monitoring framework where end-to-end probing-based solutions are used to adapt the at-a-point monitoring tools. We present a systematic framework to use probes for adjusting monitoring levels and demonstrate the effectiveness of the proposed solution using real-world examples as well as simulations.**

I. Introduction

A defining characteristic of today's information age is our reliance on data centers. The scale and complexity of these data centers has been increasing rapidly. This in turn is limiting our ability to understand and control the data center operations. To understand inherent complexity of data centers, we studied the data center of a US-based investment bank hosting hundreds of DB2 applications, firing millions of queries daily and accessing a complex array of thousands of storage devices. Such systems need automated solutions for performance and capacity management to better understand and control their operations.

A quintessential requirement for analysis of these systems is the availability of good quality of monitoring data. It is very important to systematically deploy a monitoring framework to capture behavioral characteristics of all compute, communication, and storage components. Monitoring metrics need to be collected at various layers ranging from the hardware metrics such as CPU and memory utilization, to operating system and virtual machine layers, to database and application layers, among others. A lot of monitoring tools exist that can monitor a wide variety of system components. Most of these tools are capable of monitoring a large range of metrics and can be configured as per needs. However, the quality of the monitored data is often a suspect. Effective use of the monitoring tools demands an understanding of monitoring requirements which system administrators most often lack. Instead of a well-defined process of defining monitoring strategy, system administrators adopt an ad-hoc, manual, and intuition-based approach. This leads to inconsistent and inadequate data collection and retention policies.

An aggressive or a conservative approach for monitoring can lead to either very large or very small volumes of monitoring data respectively. Very large amount of monitoring data is difficult to store and analyze. Very large volume also carries the risk of escaping the buried interesting insights. On the other hand, small amount of monitoring data carries the risk of not capturing events of interest.

The contribution of this paper is to replace the ad-hoc, manual, intuition-based approach with a more systematic, automated, and analytics-based approach for system monitoring. The key idea of the proposed approach is to use probes to understand the end-to-end performance, and to infer the criticality of the probed components from the insights gained from these probes. The inferred criticality is then used to provide dynamic monitoring guidelines for these components. A poor performing component needs monitoring at a finer-level such that many metrics are collected at a high sampling rate. The nodes can be protected from any adverse effects of increase in sampling rate by (a) using in-built OS monitoring utilities deigned to have minimal CPU foot-print and (b) enforcing a threshold on the maximum increase in sampling rate. On the other hand, for a healthy component a very low sampling rate may be adequate. We thus propose an adaptive monitoring framework where end-to-end probing-based solutions are used to adapt the at-a-point monitoring tools. Note that per-node monitoring does not capture the end-to-end view and hence monitoring alone is not enough to detect and localize faults. End-to-end probing thus complements the at-a-point monitoring solutions.

Traditionally both active probing and passive monitoring have been used in isolation. Passive monitoring techniques compute at-a-point metrics and can provide fine-grained metrics, but are agnostic to the end-to-end system performance. On the other hand, probing-based techniques compute end-to-end metrics but lack in-depth view of a component. In this paper, with an example of adaptive monitoring, we focus on how these two techniques can be used to complement each-other to develop effective solutions.

In the next section (Section II) we formulate the problem space. The following sections (Sections III, IV, V) present the proposed solution for the identified subproblems. Section VI presents a detailed experimental evaluation of the proposed approach on both real-world as well as simulation data. We

978-3-901882-48-7

discuss related work in Section VII and conclude in Section VIII.

II. Problem formulation

The probes and monitoring agents are chosen with the objective of detecting and debugging performance problems in a data center. Similar solutions can be built for other domains such as reliability, availability, and capacity management by selecting appropriate set of probing and monitoring metrics.

Monitoring Level of a component: We first introduce the notion of *monitoring level* of a component. Consider an example of a logical partition (LPAR) of mainframe box. The standard Type 70 log can capture 1640 metrics for an LPAR. However most often the experts capture only a few representative performance metrics. A larger set of metrics is captured and analyzed only in the event of component being a performance bottleneck.

Another variable in the monitoring process is the frequency of monitoring - which defines the rate at which the metrics are collected. For instance, Oracle can be configured to periodically generate AWR (Automatic Workload Repository) reports. For lack of proper monitoring strategy, these reports are most often produced for every 24 hours to capture daily statistics.

We include these two monitoring variables viz., the number of metrics and the frequency of data collection in the definition of monitoring level. A monitoring policy of a component is defined by a range of monitoring levels, where the levels vary according to the number of collected metrics and the rate of data collection. These levels can be defined with the help of expert knowledge and domain understanding. Many commercial solutions can be configured to operate monitoring at different levels.

Adapting monitoring level of a component using probes: The selection of monitoring level cannot be manual and ad-hoc, and instead needs an automated and systematic approach. In order to automatically generate monitoring level recommendations and to adapt the monitoring levels, we propose to use probing. Probes can be either on-going traffic or synthetic transactions send by probe stations [8], [13]. Probe results can be analyzed to infer health and activity on different probe paths. Probe results can thus provide an estimate of the criticality and health of different components on the probe path - which can then be used to provide monitoring recommendations.

Based on these concepts, we propose the following solution approach for adaptive monitoring: Probe stations are deployed at select nodes in the system. Monitoring agents are deployed at all nodes in the system that need to be monitored.

Actions at probe stations:

- Probe stations periodically send a few chosen probes and receive probe response.
- Probe stations analyze probe responses to infer health and criticality of nodes on each probe path.
- Probe stations maintain a rule-book to translate probe analysis into the recommendation for monitoring levels for the nodes covered by the probe path.

Actions at monitoring agents:

- Monitoring agents periodically requests for the monitoring level recommendations from the probe stations.
- Based on the recommendations from one or more probe stations, the monitoring agents compute a derived monitoring level.
- Monitoring agents adjust the monitoring policy based on the derived recommendation.

In order to realize the above solution of adaptive monitoring, following problems need to be solved. *(i)* How to select right set of probes such that criticality of all components can be correctly estimated? (Section III). *(ii)* How to analyze probe results to infer health of components on probe-path and recommend monitoring levels? (Section IV). *(iii)* How to adapt the monitoring level of a node based on the recommendations provided by one or more probe stations? (Section V).

III. Probe selection

Probes refer to test transactions such that success or failure of probes depends on the heath of the nodes through which they pass. Probes can be specially designed transactions or simply the existing on-going transactions. The approaches presented in this section require knowledge of available probe set and the nodes covered by each probe. For clarity we assume that the paths taken by the probes do not change dynamically.

The probe paths should be selected such that criticality of each node can be correctly inferred. In order to meet this objective, various criteria drive the process of probe selection.

Let the data center components being monitored are represented by the set $N = \{n_1, n_2, \ldots, n_k\}$. The set of available probes is represented by $P = \{p_1, p_2, \ldots, p_l\}$. The problem of probe selection is to identify a subset of probes $P_{sel} \subseteq P$ such that following conditions are met:

1) All nodes in the network should be covered. In other words, every node must have at least one probe passing through it. $\bigcup_{p \in P_{sel}} (nodes(p)) = N$, where $nodes(p)$ represents the set of nodes through which probe p passes.
2) One probe passes through multiple nodes and hence is not capable of uniquely identifying the health of each node on its probe path. Consequently, each node should be probed by multiple probes so that more accurate estimate of node's health can be made based on the analysis of multiple probe-paths. The average number of probes probing each node, $(\sum_{p \in P_{sel}} |nodes(p)|)/N$, should be maximized.
3) In a probe passing through a large number of nodes, one node makes a small contribution in the overall end-to-end performance of a probe. Performance degradation of one node is thus not very effectively visible in the end-to-end performance of the probe. Hence, probes passing through large number of nodes are not good candidates for providing monitoring recommendations. The average length of a probe, $(\sum_{p \in P_{sel}} |nodes(p)|)/|P_{sel}|$, should be minimized.
4) Finally, the probes come with a cost of increased probe traffic in the network and increased processing at the nodes. In order to minimize the probe traffic, the number of selected probes, $|P_{sel}|$, should be minimized.

Finding an optimal solution can be proved to be NP-Complete by reducing the Minimal Test Collection problem to this problem [12]. We next propose two approaches to solve this problem - (a) a heuristic-based greedy approach, (b) a linear programming based approach.

A. Greedy approach

In this approach we compute the weight of each probe and iteratively select probes with maximum weight until all nodes are covered. The weights are computed as follows:

Node weight: Weight of a node n_i, $Weight(n_i)$, is the inverse of the number of probes probing a node. Thus, if k probes (including the new probe) pass through a node then the weight of node is computed as $1/k$. This weight criteria is used in probe selection to give priority to less covered nodes (condition 2).

Probe weight: The weight of a probe is calculated by dividing the sum of the weights of all nodes covered by the probe with the total number of nodes covered by the probe. Thus, $Weight(p) = \sum_{n \in nodes(p)} Weight(n_i)/|nodes(p)|$. The numerator gives preference to coverage of less-covered nodes, while the division with $|nodes(p)|$ ensures that short probes are preferred over long probes (condition 3).

The set of selected probes is initialized to an empty set. In each iteration a probe with maximum weight is selected (condition 4) and added to the set of selected probes. Multiple iterations of probe selection are executed until all nodes in the network are covered (condition 1).

B. Linear programming based approach

We use the standard Set-Cover LP formulation with slight modification to incorporate the desired constraints. The algorithm using linear programming essentially includes the four stated objectives in form of constraints and objective function. It tries to minimize objective function while satisfying constraint equations. We formulated the following linear program for the problem:

$$min \sum_{p \in P} x_p \times C(p)$$

$$s.t. \sum_{p \in P} (x_p \times p_n) \geq c \quad \forall n \in N \quad ; \quad x_p \in \{0,1\} \quad \forall p \in P$$

In the formulation, x_p is a binary variable indicating whether probe p is selected ($x_p = 1$) or not ($x_p = 0$). $C(p)$ is the cost associated with probe p where cost is defined as the length of probe. Thus, $C(p) = nodes(p)$. p_n is the variable indicating presence of node n in path of probe p. p_n is 1 if node $n \in nodes(p)$ and 0 otherwise.

The objective function tries to minimize the cost of all selected probes. It satisfies the third and the fourth requirements of minimum length and minimum number of probes respectively. The constraint enforces that each node n in probe p should get probed at least c times, thereby satisfying the second requirement (average coverage). The value of c can be tuned based on the system requirements.

The above linear program is Integer Linear Program (ILP). ILP is known to be NP complete. But each probe should be either picked or left out, so we need integer solutions only. Thus if we first obtain a fractional solution to this problem and then apply rounding algorithm then an integer approximation can be obtained. To get fractional solution, we relax constraint of $x_p \in \{0,1\}$ by $1 > x_p > 0$. Now the program is solvable in polynomial time and we get fractional solution which is a set containing fractions by which each probe is picked. We apply randomized rounding algorithm to this fractional solution and get integer solution.

IV. Probe analysis

On each selected probe path, probes are sent periodically and the probe results are stored at the probe station. Consider, for instance, that the probe results capture the end-to-end latency observed on the probe path. The probe station thus captures a time-series of the latency values for each traversed probe path. The probe station performs following two analysis on the time-series: (1) The time-series is analyzed to infer the health of probe path and identify any changes in the probe performance. (2) The probe path health is translated into monitoring level recommendations for the nodes on the probe path by maintaining a rule-book. In this section, we discuss the details of these two steps.

A. Analysis of probe results

As explained above, a probe station collects probe results for a probe path and maintains a time-series of path latency values. The probe results are analyzed periodically after a fixed time window to observe any change in the end-to-end performance of the probe path. The time window can be set based on the observed failure rate of the system where a shorter time window can be set for systems with higher failure rate in order to pro-actively tune monitors. In order to keep the operations computationally light-weight, we analyze the probe results of consecutive time-windows by applying following two simple functions:

1. Deviation from normal behavior (DFN): This function captures the abnormality in the probe response. The expected value of an end-to-end metric, T_{nor}, can be obtained by domain knowledge (e.g. Service Level Agreements) or by analyzing the historical data. In order to cater to variations in the steady state of the system, T_{nor}, should be computed on a periodic basis at every significant change in the steady state of the system. Keeping the expected value T_{nor} as a benchmark, we compute the deviation of the observed value T_{cur}, as the absolute difference of the two values. Thus, $Dev(T_{cur}) = abs(T_{cur} - T_{nor})$.

2. Rate of change (ROC): This function provides insights into the severity of change and the likely future pattern of change. For instance, the rate of change can differentiate between a linear increase and an exponential increase in the latency. Given the value T_{cur} observed in the current window and the value T_{prv} observed in the previous window, we compute the rate of change as: $Rate(T_{cur}) = abs(T_{cur} - T_{prv})/T_{prv}$.

Both ROC and DFN complement each other in analyzing probe results. While ROC captures the rate of change, DFN provides a measure of the severity of changed values. A gradual change that is not prominently visible in consecutive time-windows may go unnoticed by ROC, whereas DFN can successfully capture when such gradual change crosses the acceptable limits.

Above analysis of probe results can be further extended to include various extensions such as: Instead of a fixed time-window, the window size can be adapted based on probe health. Similarly windows can be made overlapping to avoid adverse effect of noise and transient fluctuations. The solution can be further enhanced by capturing expert knowledge to infer the meaning of failures of specific domain-dependent probes.

B. Translating probe analysis into monitoring level recommendations

We next use the values for deviation from normal behavior (DFN) and rate of change (ROC) to propose a monitoring level for all nodes on the probe path. We propose a rule-book to map the values of DFN and ROC to monitoring level. Each probe station uses the rule-book to maintain a monitoring level recommendation for each probe path that it traverses.

1) Rule-book creation: The rule-book is created using the historical data of probe results (e.g. probe latency) as follows:

1) Divide the historical data into time-windows as done in probe analysis step (Section IV-A).
2) Compute DFN for each pair of consecutive windows.
3) Group the values of DFN into clusters using BIRCH algorithm [20]. The number of clusters is kept same as the required number of monitoring levels. Note that the number of monitoring levels depends on the levels supported by the monitoring tools.
4) These clusters thus form the rule-book entries, where each cluster represents a monitoring level. The rule-book thus maps a range of DFN values (belonging to a cluster) to a monitoring level.
5) Repeat the above steps to construct rulebook for ROC values.

2) Rule-book lookup and update:

1) Given a value of DFN of the current window, use BIRCH algorithm to lookup the DFN rule-book and find the cluster to which this value belongs. The monitoring level of the identified cluster is recommended as the suitable monitoring level.
2) Similarly, given a value of ROC of the current window, lookup the ROC rule-book to derive monitoring level.
3) The BIRCH algorithm adapts the rule-book by incrementally re-clustering the DFN and ROC values based on the observed new values.

3) Deriving the monitoring level recommendation: Given the recommendations from the rule-book lookup, the derivation of monitoring level recommendation for all nodes on the probe path is now straight forward. The probe station computes the recommended monitoring level using the past monitoring level and the recommended monitoring levels from the DFN and ROC rule-book as follows:

$$ML_{cur} = \frac{ML_{prv}+ML_{DFN}+ML_{ROC}}{3}$$

This equation can be modified to assign different weights to the three monitoring level recommendations.

V. Adapting monitoring level at a node

The recommendation obtained from each probe path is assigned a confidence based on the length of the probe path. In a probe passing through a large number of nodes, one node makes a small contribution in the overall end-to-end performance of a probe. Performance degradation of one node is thus not very effectively visible in the end-to-end performance of the probe. Hence, recommendation from shorter probe is assigned a higher weight over that from a longer probe. The final monitoring level for a node n is derived by computing a weighted average of the individual recommendations.

$$ML_n = \sum_{p \in probes(n)} \frac{w_p}{w_{all}} * ML_p$$

, where $w_p = 1/|nodes(p)|$ and $w_{all} = \sum_{p \in probes(n)} w_p$

where $probes(n)$ refers to probe paths passing through node n and $nodes(p)$ refers to number of nodes through which probe p passes. As explained in Section II, this monitoring level refers to the frequency of data collection and the number of metrics collected.

VI. Experimental evaluation

A. Application in a real-world environment

We applied the proposed algorithm on the real-world data center of a US-based retail store hosting DB2 applications on mainframes. We collected monitoring logs at 78 DB2 applications (transaction logs), logical partitions (LPARs) (type 70 logs), disk volumes (type 74 logs), and datasets (type 42 logs). We collected data using both the traditional (fixed non-adaptive) monitoring and adaptive monitoring for a time duration of one week. In traditional monitoring the metrics were collected at fixed periodic time-interval of 15 mins. In adaptive monitoring the data collection frequency was adapted based on probe results. We did not perform any explicit probe selection in this experiment and instead used the ongoing DB2 transactions as probes. The monitoring logs of adaptive monitoring (AM logs) were significantly smaller than that of traditional monitoring (TM logs). The data reductions obtained in the logs of DB2 applications, LPARs, and volumes is 68.4%, 59.1%, and 68.3% respectively. Note that the AM logs are smaller in size because in normal conditions the performance degradations and the outages are fewer. In the scenarios where the outages are very frequent, the AM logs will be larger in size but at the advantage of better quality monitoring data for more accurate analysis. We next demonstrate the quality of the reduced data in capturing all events of interest. We performed various performance debugging operations and compared the findings obtained using logs collected from traditional monitoring and adaptive monitoring.

Time-series and statistics: We measured four moments, namely mean, standard deviation, kurtosis and skew, to quantitatively measure the similarity of data of both adaptive and traditional monitoring. The result for the moments were 87.35, 337.33, 52.45 and 6.86 for traditional monitoring and 95.64, 349.38, 45.10 and 6.32 for adaptive monitoring. It can be seen that the AM logs (121 data points) capture all significant changes in latency as captured by TM logs (384 data points).

Root-cause analysis using causality models: In order to debug cause behind high CPU utilization of an LPAR, we developed causality models between various workload components and resource components at DB2 applications and LPARs. We used Bayesian networks (*deal* package in *R*) to construct these models. Figure 1(a) shows the causality models constructed by AM and TM logs. It can be seen that even after 68% reduction in collected data, the AM logs are able to capture all dependencies as captured by TM logs. We used this Bayesian network to identify the likely causes behind very high CPU utilization at an LPAR ($> 80\%$). Figure 1(a) shows the hypothesis (likely cause and confidence) obtained by both logs. Both logs identify same workload components as the likely causes behind high CPU utilization.

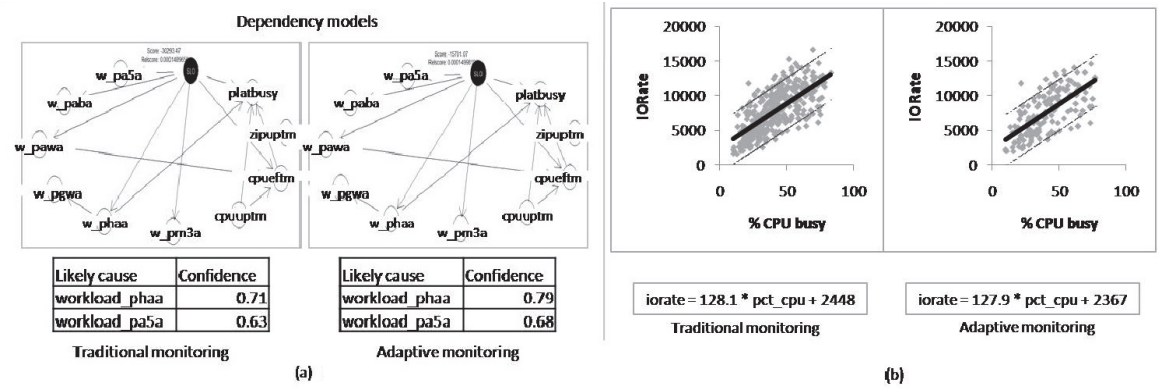

Fig. 1. Comparison of analysis results obtained from traditional monitoring and adaptive monitoring (a) Dependency model between CPU workload and likely causes of high CPU utilization, (b) regression model between workload and resource utilization and headroom analysis.

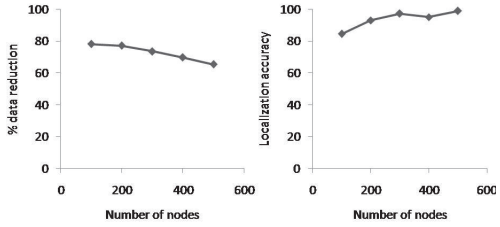

Fig. 2. (a) Data reduction and (b) fault localization accuracy for different network sizes.

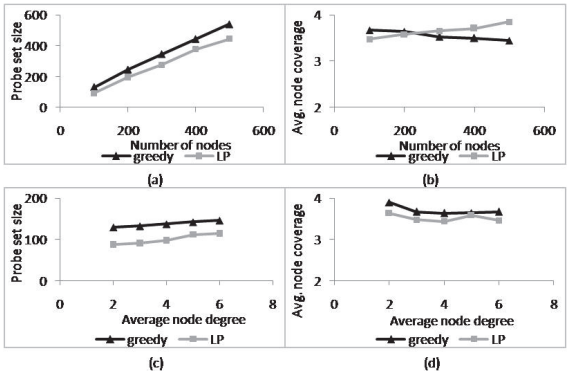

Fig. 3. Comparison of Greedy and LP-based algorithm for (a,b) varying network sizes, (c,d) varying average node degree.

Headroom analysis using regression: We computed the available free CPU and used regression models to derive additional I/O rate that can be supported with the available free CPU. Figure 1(b) shows the regression models calculated for the AM and TM logs. Instead of a single line linear model, we modeled the relationship with a linear-band in order to better capture the workload-resource relationship. Figure 1(b) also shows the available CPU headroom and the derived additional I/O-rate capacity. It can be seen that the results of AM logs are very close to the results of TM logs.

Through these experiments we demonstrate that adaptive monitoring was able to significantly reduce the monitoring logs (60% to 70% reduction) while capturing all events of interest. All performance analysis operations were successfully performed and the hypothesis produced by AM logs were similar to that produced by TM logs.

B. Evaluation on simulation data

Experiment setup: In order to perform the sensitivity analysis of the proposed approach we next present the evaluation of the proposed approach on a variety of network topologies. We generated topologies using BRITE [11]. Each node in the topology was then modeled as a physical machine using CSIM simulator [15]. Various resource metrics within a node were modeled using the Machine Repairman Model [14]. Workload was generated in the form of requests that enter from a node, pass through one or more nodes, and exit from a node. Request were designed to require some processing at the nodes through which it passes. We used the proposed greedy probe selection algorithm to select probes and captured the end-to-end latencies of the probes. Monitoring agents at each node were capturing the utilization of resource metrics modeled at each node. For each topology we collected per-node monitoring data using both traditional as well as adaptive monitoring.

Evaluation of fault localization: We introduced faults in the form of significant persistent increase in per-hop delay at randomly chosen nodes. The per-hop delay is increased by overloading one or more resources within a node. These faults resulted in increased resource utilization at one or more resources within the node. These faults also resulted in increased end-to-end latencies on the probes that passed through the faulty nodes. We then analyzed the increased end-to-end latencies and the per-node metrics to perform fault localization and identify the most likely faulty nodes. We applied the fault localization algorithms on both AM and TM logs and compared the accuracy in localizing the actual fault.

In order to perform fault localization we constructed Classification and Regression Trees (CARTs) with the latency values as the leaf nodes and the node metrics as the intermediate classification nodes. We used CARTs to localize the node

metrics that can be the most likely causes behind high probe latencies. Using CARTs we report the top most likely cause of increased latencies and the likely confidence in the proposed hypothesis. We evaluated effectiveness of adaptive monitoring using following two metrics:

$$PercentageDataReduction = \frac{DataSize_{TM} - DataSize_{AM}}{DataSize_{AM}} * 100.$$

$$LocalizationAccuracy = \frac{ResConfidence_{TM} - ResConfidence_{AM}}{ResConfidence_{TM}} * 100.$$

Experiment results: Figure 2 shows the data reduction and localization accuracy for different network sizes ranging from 100 nodes to 500 nodes. It can be seen that the data reduction percentage ranges between 60% and 80% with the accuracy in both cases ranges between 80% and 98%. With increasing network size, more probes are needed to cover the network. This results in more probes passing through a node. The recommendations thus become more conservative resulting in larger amount of data collected. With larger log size, the accuracy for both fault localization and headroom analysis tends to further increase up to 98%.

C. Evaluation of probe selection algorithms

The proposed probe selection algorithms were evaluated on (1) probe set size and (2) average node coverage. Figure 3(a,b) shows the comparison of the two algorithms for varying network sizes (number of nodes) from 100 to 500 nodes. The probe set size increases with increasing network size due to increasing demand of more nodes to cover while preferring short-length probes. With increasing network size, the average node coverage provided by LP-based algorithm increases while the average node coverage provided by Greedy algorithm decreases. In Greedy algorithm, as the number of nodes increases the coverage decreases because the algorithm stops once whole network is covered. Where as in LP approach, the coefficient on which the rounding algorithm works are very small for small network size and increases as network size increase thus rounding algorithm result improves. Thus, for larger networks the LP-based algorithm provides better node coverage with lesser number of probes as compared to the Greedy algorithm. Figure 3(c,d) compares the two algorithms for varying average node degree from 2 to 6 in a 100-node network. The probe set size increases with increasing average node degree because of the increase in the availability of short-length probes. As the number of nodes is fixed, the change in average node degree does not have a significant effect on the average node coverage. The average node coverage remains almost constant.

VII. Related work

In the past, attempts have been made to make the monitoring logs more condensed and insightful. Techniques such as Principal Component Analysis and Independent Component Analysis have used to identify and retain independent variables and remove the redundant and derived metrics [9]. Data compression techniques have also been used to define the purging and retention policies.These solutions demand high data-processing at the monitoring agents which is very often discouraged in operational systems. Furthermore, when feasible, these solutions can be used in conjunction to further improve the monitoring process. The concept of dynamically adjusting the monitoring frequency has been referred in [5] for monitoring customer QoS experience. Authors in [2] introduce the notion of a *sketch* to adapt the frequency of collection of an event based on its past rate of occurrence.

Active probing and passive monitoring have been widely used in the past for network measurement and monitoring. However, most of the past work has used them in isolation. Passive monitoring based techniques have been used for various intra-node analysis operations [4], [6], where as probing-based solutions have been widely used for end-to-end operations and analysis [18], [12], [13]. Below we discuss some attempts where the two techniques have been used in collaboration. The AIR framework [16] uses active probing to discover critical symptoms when passive reasoning is insufficient to explain the problem. Similarly the authors in [10] use hybrid methodology of monitoring and probing to compare the latency parameter for LTE and HSPA networks. They inject probes to capture delays caused by different intermediate components, thus using the two techniques in isolation. In [1] a monitoring framework called SMRM has been proposed which uses probing to generate directed multicast traffic in absence of actual transactions and collect real-time quality reports using passive monitoring. On similar lines, both off-line and on-line tools are used in [3], [19], [7] for root-cause analysis. If the off-line transaction log analysis cannot pinpoint the root cause of error, sufficient symptoms for localizing the root cause can be collected by running more transactions as proposed in [3]. In [17] CPU intensive probes are used along with OS utilities such as vmstat and uptime to collect more accurate measurements of CPU. Our paper differs from the past attempts in presenting a systematic framework to use probes for adjusting monitoring levels. We present algorithms to select and analyze probes, and to dynamically adapt the monitoring policies based on probe analysis.

VIII. Conclusion

We proposed an adaptive monitoring framework where we used end-to-end probing-based solutions to adapt the at-a-point monitoring tools. We presented solutions to select and analyze probes and derive recommendations from probe analysis to adapt monitoring levels at a node. We demonstrated the effectiveness of the proposed solution using real-world examples as well as simulations. We showed that adaptive monitoring resulted into a 60% to 70% reduction in monitoring logs without compromising into the accuracy of analysis. With an example of adaptive monitoring, we demonstrated how active probing and passive monitoring can be used to complement each-other to develop effective solutions. We believe that many powerful network management solutions can be designed by adopting a hybrid approach that uses both passive monitoring and active probing.

REFERENCES

[1] Ehab Al-Shaer and Yongning Tang. Qos path monitoring for multicast networks, 2002.

[2] Sapan Bhatia, Abhishek Kumar, Marc E. Fiuczynski, and Larry Peterson. Lightweight, high-resolution monitoring for troubleshooting production systems. In *Proceedings of the OSDI'08*.

[3] J. Gao, G. Kar, and P. Kermani. Approaches to building self healing systems using dependency analysis. In *NOMS 2004*, 2004.

[4] L.P. Gaspary and E. Canterle. Assessing transaction-based internet applications performance through a passive network traffic monitoring approach. In *GLOBECOM '04*, 2004.

[5] J. Groenendijk, Yangcheng Huang, and L. Fallon. Adaptive terminal reporting for scalable service quality monitoring in large networks. In *CNSM 2011*, Oct. 2011.

[6] Se-Hee Han, Myung-Sup Kim, Hong-Taek Ju, and James Won-Ki Hong. The architecture of ng-mon: A passive network monitoring system for high-speed ip networks. In *DSOM'02*, 2002.

[7] Ling Huang, Xuanlong Nguyen, Minos Garofalakis, and Joseph M. Hellerstein. Communication-efficient online detection of network-wide anomalies. In *INFOCOM'07*, 2007.

[8] Deepak Jeswani, Nakul Korde, Dinesh Patil, Maitreya Natu, and John Augustine. Probe station selection algorithms for fault management in computer networks. In *Proceedings of the 2nd international conference on COMmunication systems and NETworks*, COMSNETS'10, pages 357–365, Piscataway, NJ, USA, 2010. IEEE Press.

[9] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *SIGCOMM'04*, 2004.

[10] Markus Laner, Philipp Svoboda, Peter Romirer-Maierhofer, Navid Nikaein, Fabio Ricciato, and Markus Rupp. A comparison between one-way delays in operating hspa and lte networks. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2012 10th International Symposium on*, pages 286 –292, may 2012.

[11] A. Medina, A. Lakhina, I. Matta, and John Byers. Brite: An approach to universal topology generation. In *MASCOTS'01, Cincinnati, Ohio*, Aug. 2001.

[12] M. Natu and A. S. Sethi. Probabilistic fault diagnosis using adaptive probing. In *DSOM 2007, San Jose, CA*, Oct. 2007.

[13] M. Natu and A. S. Sethi. Application of adaptive probing for fault diagnosis in computer networks. In *NOMS'08*, Apr. 2008.

[14] Allan Scherr. Machine repairman model of mit time sharing system, 1965.

[15] Herb Schwetman. Csim: a c-based process-oriented simulation language. In *WSC '86*, 1986.

[16] Y. Tang, E. S. Al-Shaer, and R. Boutaba. Active integrated fault localization in communication networks. In *IM 2005*, pages 543–556, May 2005.

[17] Rich Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Perform. Eval. Rev.*, 30(4):41–49, March 2003.

[18] Likun Yu, Lu Cheng, Yan Qiao, Yiguo Yuan, and Xingyu Chen. An efficient active probing approach based on the combination of online and offline strategies. In *CNSM'10*, pages 298 –301, oct. 2010.

[19] Minlan Yu, Albert Greenberg, Dave Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. Profiling network performance for multi-tier data center applications. In *NSDI'11*, 2011.

[20] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *SIGMOD'96*, pages 103–114, 1996.