A Flexible Architecture Integrating Monitoring and Analytics for Managing Large-Scale Data Centers

Chengwei Wang, Karsten Schwan, Vanish Talwar*, Greg Eisenhauer
Liting Hu, Matthew Wolf

Center for Experimental Research in Computer Systems
College of Computing, Georgia Institute of Technology, Atlanta, GA 30318, USA
{flinter, schwan, eisen, liting, matthew.wolf}@cc.gatech.edu

* HP Labs, Palo Alto CA 94306, USA
vanish.talwar@hp.com

ABSTRACT

To effectively manage large-scale data centers and utility clouds, operators must understand current system and application behaviors. This requires continuous, real-time monitoring along with on-line analysis of the data captured by the monitoring system, i.e., integrated monitoring and analytics – Monalytics [28]. A key challenge with such integration is to balance the costs incurred and associated delays, against the benefits attained from identifying and reacting to, in a timely fashion, undesirable or non-performing system states. This paper presents a novel, flexible architecture for Monalytics in which such trade-offs are easily made by dynamically constructing software overlays called distributed computation graphs (DCGs) to implement desired analytics functions. The prototype of Monalytics implementing this flexible architecture is evaluated with motivating use cases in small scale data center experiments, and a series of analytical models is used to understand the above trade-offs at large scales. Results show that the approach provides the flexibility to meet the demands of autonomic management at large scale with considerably better performance/cost than traditional and brute force solutions.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Network Operating Systems*

General Terms

Management, Design, Experimentation

Keywords

Monitoring, Analytics, Data Center, Cloud

1. INTRODUCTION

Monitoring and data analysis[1] are two fundamental elements of data center management. Monitoring tracks desired hardware and software metrics. Analysis evaluates these metrics to identify system or application states for troubleshooting, resource provisioning, or other management actions. To effectively manage modern data centers, both monitoring and associated analytics must be performed in real-time and at scales of tens of thousands of heterogeneous nodes with complex network and I/O structures.

Previous research on monitoring has created scalable methods for real-time data collection and aggregation [13, 40, 37, 33, 31], to support efficient on-line queries that answer questions like 'which machines have CPU utilization above 90%?'[25]. 'Analysis-focused' research has drawn from areas like data mining, machine learning, and statistics to create techniques that assist in or automate problem diagnosis [5, 6, 10, 11, 15], with high accuracy and significantly reduced human intervention. However, while monitoring has been shown feasible at scale and in real-time, analysis is typically performed after a volume of monitoring data has been written to disk-resident logs, or in a central location, which impedes the scalability of on-line monitoring and analysis tasks. Further, due to lack of underlying infrastructure support, analytics often require global data – over time and space – making it difficult to use them on-demand and in real-time. Finally, in modern virtualized utility or cloud computing systems, operators or administrators have limited visibility into the virtual machines running on data center machines. This prevents them from using problem diagnosis methods that require such insight.

To address these challenges, we propose a system integrating monitoring with analytics, termed *Monalytics*, which can capture, aggregate, and incrementally analyze data on-demand and in real-time, only where (i.e., in situ) and to the extents needed by intended management actions. This is first introduced in [28], with initial results indicating that Monalytics should be built to respect notions of 'scope' in time and space for (i) acceptable overheads, and (ii) appropriate delays between when certain conditions arise and when they are detected (and thus, can be acted on). To

[1]We use 'analysis' and 'analytics' interchangeably in this paper.

do so, however, requires a flexible architecture to accommodate the changing and diverse characteristics of analytics, with cost-effectiveness in large-scale data centers.

This paper presents the design and evaluations of Monalytics' flexible architecture built upon dynamic *distributed computation graphs* (DCGs), providing the following technical contributions:

- `Pro re nata (PRN) deployment`: an important property of Monalytics is its instantiations of analytic functions only where and when they are needed. In other words, Monalytics must have capabilities for dynamically *zooming in* to 'interesting' locations and periods of time. Such capabilities can also benefit scalability by substantially reduced costs compared with systems forced to 'watch everything all the time'. *We validate the PRN deployment in two realistic use cases and compare them with traditional brute-force approaches.*

- `Reducing 'Time to Insight' (TTI) and cost`: a vital metric for assessing the performance of monitoring/analysis actions is *Time to Insight* (TTI) capturing the total delay between when 'interesting' events occur until they are recognized (i.e., after analysis is complete). Using this metric and also assessing the costs incurred along with different values of TTI, *we evaluate the cost-effectiveness of alternative topologies used to construct DCGs, and validate our novel flexible hybrid DCG design.*

The evaluations are based on both experimental results and performance modeling at scale. They show that, as the key to autonomic data center management, the flexible PRN deployment of DCGs enables continuous operation at scale and is cost-effective in attaining TTI required for various analytics functions. Compared with brute force solutions and traditional static topologies, Monalytics yields up to 92% TTI reduction and 86% lower cost.

The remainder of this paper is organized as follows. Section 2 elaborates challenges for integrating monitoring and analysis in large-scale data centers. The two use cases driving our research are presented in Section 3. The design of DCG-based Monalytics software is described in Section 4, and a series of analytical models assessing DCG topologies is detailed in Section 5. Section 6 presents experimental and analytical evaluation results, related work appears in Section 7. Conclusion and future work are in Section 8.

2. PROBLEM STATEMENT

Accommodating the Variety of Analytics: A first step towards integrating monitoring and analytics is to recognize that analytics approaches vary widely in terms of computational complexity and implementation (e.g., centralized or distributed, etc.). As an illustration, we list representative analytics approaches for data center management in Table 1. The second column describes core algorithms, including simple sorting or traversal algorithms, machine learning methods (e.g., TAN–Tree Augmented Naive Bayes and K-Clustering), and algorithms used for specific purposes, such as Convolution [6] and Pathmap [5]. The computational complexities of these various analytics approaches range from linear to exponential, with differences in their computation styles as well. For instance, *MAX* and *Top-K* could be run in a distributed manner using an aggregation tree, while TAN Bayes is processed at a centralized location. Magpie

Table 1: Typical Analytics Approaches. N:number of monitoring samples, n:number of metrics in each monitoring sample, k:number of centroids, Δ:increment number of samples, $\{p, e, S, E, W, \tau, m\}$:approach-specific parameters

Name	Core Algorithms	Computational Complexity
MAX [37]	Traversing	O(nN)
Top-K [37]	Sorting	O($nNlogN$)
Signatures [16]	TAN Bayes	O(n^2N) [19]
	K-Clustering	O($N^{nk+1}logN$) [23]
Pinpoint [15]	UPGMA Clustering	O(N^2) [32]
Magpie [11]	Incremental Clustering	O($n(N+\Delta)$) [12]
Pranaali [27]	TAN Bayes	O(n^2N) [19]
Black-Box [6]	Nesting Algorithm	O(Np)
	Convolution Algorithm	O($em+eSlogS$)
E2EProf [5]	Pathmap	O($E[\frac{W}{\tau}]^2$)
Sherklock [10]	Inference Graph	O(3^m)

can use incremental clustering as new monitoring data arrives, whereas [6] needs to collect traces for some period of time and do a one-time off-line analysis.

A challenge resulting from this variety is *how to accommodate such a wide range of analytics for autonomic management in large scale data centers?* Previous research does not address this challenge when designing monitoring or aggregation systems, instead focusing optimization efforts on the communication overheads incurred for monitoring, including message volumes and bandwidth consumption, delays in data delivery, etc., using techniques such as in-network processing [30] and source-based filtering [20, 18, 24]. Section 6 shows that traditional static system designs cannot easily accommodate analytics functions of varying computation complexities.

Meeting the Varying Requirements in Time and Space: Autonomic management requires *on-demand* monitoring and analytics to diagnose problems and to understand system behavior, so as to take timely corrective actions to meet service level objectives. The actual analytics used, however, and the level of detail at which they must operate *vary* over time, across the different components or entities being managed, as well as across the multiple levels of abstraction present in complex data center systems. Specifics for this depend on the types of applications or services deployed, system loads, hardware configurations used, desired service level objectives, etc. In any large scale data center, therefore, there will be heterogeneous and non-uniform monitoring and analytics needs over time and space, and these properties also and perhaps, even more so, hold for cloud environments. The elasticity provided by cloud environments allow system deployments to scale-up and scale-down, thereby directly affecting associated analysis structures and requirements.

The resulting challenge for a Monalytics system is: *how to design a system capable of dynamically configuring monitoring and analysis structures to meet the varying requirements in time and space?* Section 6 shows that static solutions do not meet the flexibility needed at large scale, whereas our Monalytics design can address the challenge with significant potential performance improvements.

Improving Cost-Effectiveness: With active management of data center hardware and applications becoming increasingly common, it is important to study the costs of management [36, 29]. This should include budgeting both for the capital costs of management hardware resources and for their operational costs (power, cooling, administration, etc). Such budgeting must differentiate costs incurred when management uses *dedicated* hardware (e.g., HP's iLO [2] or IBM's Director [4]) versus when it is *collocated* with the machines being monitored. Intuitively, the former typically provides performance benefits at higher capital costs, whereas the latter has less capital costs but offers reduced performance since it competes for machine resources with the applications being run in the data center.

A resulting challenge is: *how to design a cost-effective system minimizing management cost while yielding the best possible performance?*. As shown in Section 6, when considered at scale, high performance for management may incur potentially prohibitive capital cost. Our research addresses this by explicitly modeling the capital costs of management resources. Preliminary results show that Monalytics can outperform static systems at considerably lower capital costs.

3. MOTIVATING USE CASES

3.1 ZIA: Zoom-In Analysis in Large Scale Internet Services

Automatically inferring causal dependencies between components[6, 10, 5] – causal path inference – is known to be useful for identifying performance problems in multi-tier web applications. A typical example is the detection of bottlenecks in the end-to-end latencies experienced by requests. However, the monitoring requirements – large request traces – and high computational complexity make it expensive and most likely, unrealistic to determine all causal paths in large scale data centers, whether they contribute to bottlenecks.

Consider a flexible monitoring and analytics (Monalytics) software that can address this issue by implementing a two-phase approach that evaluates only those dependencies that likely contribute to sudden bottlenecks. This PRN method, termed Zoom-in Analysis (ZIA), allows system administrators to use powerful tools like those required for causal path detection at large scales. Key to the implementation of ZIA is the ability to deploy overlays and analysis functions on any subset of machines, at runtime and as needed (PRN) for further diagnosis.

The phases of ZIA are described with an illustration in Figure 1a, showing a practical use case. The first phase is *global light-weight anomaly detection.* Anomaly detection runs continuously with a Monalytics overlay (i.e., the DCG elaborated in Section 4) that spans all appropriate data center machines. The overlay collects application-level SLO (Service Level Objective) metrics and system metrics, e.g. transaction response times and CPU utilizations, and it processes them with a low-cost algorithm described in [38, 39]. For simplicity, the figure depicts a hierarchical overlay, but other topologies may also be used. Anomaly detection has each processing unit aggregate and analyze its local metrics and then pass results to its parents. The root has a global view of all machines.

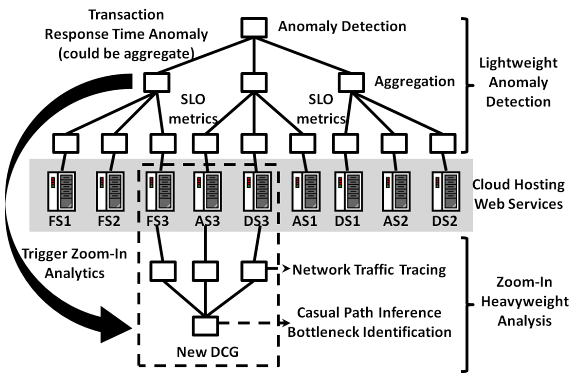
Alarms are raised if anomalies in, say SLOs (e.g., long request duration) are detected, which then triggers the second phase of ZIA. The second phase performs *in-depth analysis in 'problem areas'*, which touches only upon a subset of the machines being monitored, i.e., those for which anomalous behavior has been observed. As illustrated in Figure 1a, this involves creating a new overlay at runtime and on-demand to cover the machines associated with Service 3, Front-end Server 3 (FS3), Application Server 3 (AS3), and Database Server 3 (DS3). This overlay uses a network tracing facility to identify all messages between these servers (RPC requests/replies, IP packages). The data is sent to an analyzer for casual path inference, using an inference engine like the one described in [6]. Knowledge about causal paths can then help identify the potential bottlenecks that caused large end-to-end transaction response times.

The concept of zoom-in analysis generalizes to other problems and applications, and in fact, we next show a second use case in which zoom-in is performed in an entirely different fashion. The effect of using PRN techniques like zoom-in, of course, is that intensive or high overhead analytics can be focused on likely problem areas instead of entire systems. This substantially reduces monitoring overheads and interference to applications compared to a brute force approach that performs causal path inferences all the time.

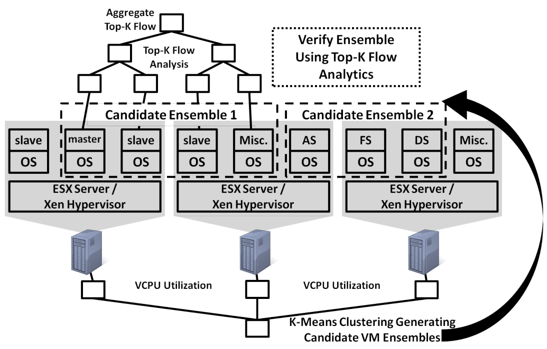
3.2 VMC: Virtual Machine Clustering

A common problem experienced in data centers and utility clouds is lack of knowledge about the mappings of the services being run by or offered to external users to the sets of virtual machines (VMs) that implement them. This makes it difficult to manage VM ensembles – sets of VMs implementing some common service – to attain provider goals like minimizing the resources consumed by certain services or reducing the power they consume on data center machines. A case in point is high network resource consumption when in a public cloud, a VM ensemble running a Hadoop Mapreduce application [1], for instance, is deployed on physical hosts located on different racks rather than on the same rack. This substantially reduces the cross-section bandwidth available in the data center because typically, hardware is configured in ways that offer much less bandwidth across versus within racks. The outcomes are not only deteriorated performance for the Hadoop application, but also negative impacts on other applications running on these machines. The problem is important because cross-section bandwidth is a limited commodity and increasing it can be costly in terms of the network switches and routers that must be purchased.

Key to properly placing VM ensembles is to first recognize their existence, i.e., to identify them, but this can be difficult. First, system administrators running utility clouds typically have limited knowledge about the applications being run, in contrast to the previous ZIA use case in which we assume an a priori knowledge about the web application's configuration. As a result, administrators must use black-box methods to identify VM ensembles. One such method, based on correlation analysis and described in [9], is efficient in terms of the overheads being experienced, thereby permitting the continuous monitoring necessary to deal with dynamic changes in ensembles and their dynamic arrivals and departures commonly seen in utility data centers. Unfortunately, while overheads are low, the method only discovers *potential* VM ensembles, thus making it necessary to use additional monitoring to distinguish potential from actual ensembles. With Monalytics, such additional, in-depth analysis can be done where and as needed (PRN) and at runtime. Specifically, the analysis method we use is one that inspects all data exchanged between the VMs in a candidate ensemble, both to confirm the ensemble's existence and to gain additional information for improved ensemble placement, such as inter-VM message volumes. As will be explained in later sections, this PRN method is implemented by installing net-


Transaction Response Time Anomaly (could be aggregate)
Anomaly Detection
Aggregation
SLO metrics
SLO metrics
Lightweight Anomaly Detection
Cloud Hosting Web Services
FS1
FS2
FS3
AS3
DS3
AS1
DS1
AS2
DS2
Trigger Zoom-In Analytics
Network Traffic Tracing
Casual Path Inference Bottleneck Identification
New DCG
Zoom-In Heavyweight Analysis

(a) Use case I: Zoom-In Analysis (ZIA)


Aggregate Top-K Flow
Top-K Flow Analysis
Verify Ensemble Using Top-K Flow Analytics
Candidate Ensemble 1
Candidate Ensemble 2
slave
OS
master
OS
slave
OS
slave
OS
Misc.
OS
AS
OS
FS
OS
DS
OS
Misc.
OS
ESX Server / Xen Hypervisor
ESX Server / Xen Hypervisor
ESX Server / Xen Hypervisor
VCPU Utilization
VCPU Utilization
K-Means Clustering Generating Candidate VM Ensembles

(b) Use case II: Virtual Machine Clustering

Figure 1: Two Representative Use Cases of Monalytics. The group of servers in the middle of Figure (a) exemplifies a utility cloud that hosts Internet services. FS[i], AS[i], DS[i] denote Front-end Server, Application Server, and Database Server for Service i, respectively. The assemblies of rectangles in Figures (a) and (b) represent autonomic deployments of Monalytics overlays. In Figure (b), each of the three sample servers hosts three virtual machines, where slaves are the worker processes in the mapreduce application, controlled by the master process. Virtual machines running applications other than mapreduce or Internet services are named Misc.

filter modules only into appropriate machines and building an appropriate DCG to analyze the data captured in this fashion (see [9, 22] for additional detail).

A small-scale use case is depicted in Figure 1b, in which a basic overlay is deployed on three sample servers hosting nine VMs running multiple applications, including a Mapreduce application and a multi-tier web service code. The CPU utilizations of VMs (i.e., VCPU utilizations) are collected on each host. A central node gathering the data runs the lightweight clustering algorithms described in [9]. Its output is a list of potential VM ensembles, i.e., VMs that probably communicate regularly. An ensemble spanning different racks is picked as a potential target for VM migration, but before constructing a migration plan and carrying it out, a PRN method is used to assess the actual amounts of traffic they exchange. This involves creating a new overlay, on-demand, targeting only this ensemble of VMs (a master VM, two slave VM, and a Misc. VM). The new overlay gathers IP package statistics from the VMs and analyzes total network traffic by using *Top-K* flow analysis [26]. The analysis finds the k flows that most contribute to the traffic between any two VMs and their sizes. It eliminates any member of the ensemble with coincidental correlations in terms of CPU usage, and provides the flow data needed to better assess the cost-benefits derived from VM migration.

4. SYSTEM DESIGN

The building block of the flexible Monalytics architecture is the Distributed Computation Graph (DCG), a reconfigurable overlay that undertakes monitoring data collection, exchange, and processing. As illustrated in Figure 2, a DCG is comprised of two types of basic entities, Data Collector(DC) and Monitoring Broker (M-Broker). A DC typically runs on the monitored node, invoking general monitoring tools to collect run time states. Depending on analysis needs, monitoring metrics can be gathered periodically as samples, continuously as traces, or in one-shot. DCs send them to the M-Brokers to which they are attached, using local shared memory when the DCG is collocated with monitored nodes (i.e., the collocated mode) or via the network when the DCG is deployed in dedicated management hardwares (i.e., the dedicated mode). M-Brokers are also connected to each other through a topology for cooperative analysis. A DCG achieves the flexibility needed for scalable, autonomic management through three design features:

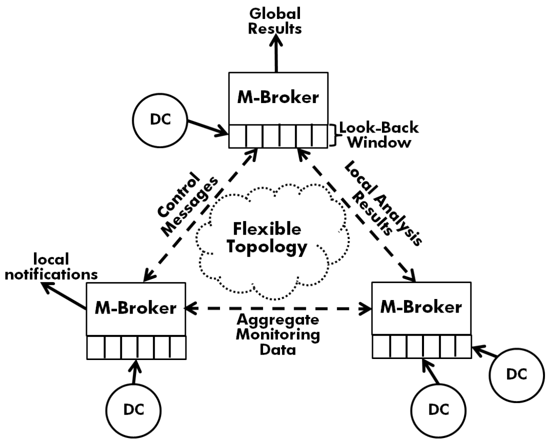


Figure 2: A sample DCG with three M-Brokers and four data collectors (DCs). DCG controllers illustrated in Figure 3 can construct and manage DCGs dynamically.

Flexible Analytics Containers (M-Brokers): An M-Broker is the key analytics processing unit in Monalytics. It aggregates the raw data, and passes aggregate results to other M-Brokers for further aggregation for larger scope, or into global states, or for cooperative analysis. Specifically, current M-Brokers maintain analysis state structured as Look-Back Windows (LBWs) supporting on-line analysis. They use these windows to store the monitoring data, to aggregate data, or to maintain intermediate analytics results for some period of time. Analysis actions operate on LBWs, and they are updated as new data flows in. A M-Broker can be deployed in collocation with the node monitored or on dedicated management components such as management blades and processors in the data center. As an analytics container, an M-Broker is able to hold various kinds of analytics functions with associated DCGs. In other words, multiple DCGs implementing different analytics may be constructed by the same set of M-Brokers and DCs.

On-Demand DCG Creation (DCG Controllers): An important attribute of Monalytics is that DCs, M-Brokers, and DCGs are dynamic entities which can be created, connected to each other, and terminated on-the-fly. It is in this fashion that new analytics functions can be initiated and stopped on demand, and existing functions can be adjusted. This on-demand creation and management is done by *DCG controllers*. A controller can create M-Brokers or DCs on any node to which it has access, or reuse existing ones. After that, it connects DCs and M-Brokers using a DCG topology that could be predefined by users or auto-generated based on given management policies. As soon as the DCG is constructed, the DC will start pushing monitoring data into DCG and M-Brokers will process it. The DCG can be terminated via signals issued by the controller, or that task can be assigned to some M-Broker. The controller also tracks the states of DCGs.

For example, in the ZIA use case described in Section 3.1, we initially create DCs on each host to collect application-level monitoring data through JMX and system level metrics using Syststat. Those DCs are attached to M-Brokers that are created and constructed into a hierarchy. Each M-Broker is equipped with the EbAT [38, 39] functions for anomaly detection. When an anomaly is caught, a new DCG is created with DCs on the suspicious nodes and a single M-Broker. The formers trace network flows and relay the data to the latter, which uses a causal path inference engine [6] to analyze the data. When the causal inference process finishes, the DCG is terminated, and the associated DCs and M-Broker can be reclaimed.

Flexible Topologies: The third aspect of the DCG is its ability to use multiple topologies across M-Brokers to meet various analytics requirements in the data center, and to also meet various performance and cost needs. These topologies include the traditional ones used in previous monitoring/aggregation systems. In addition, we propose what we term *hybrid DCGs* that consist of heterogeneous topology structures adapted to different regions of the data center, and interconnected through an inter-region topology. Each region can have its own local topology among the M-Brokers, and the leader M-Brokers of respective regions can be interconnected with another topology. This is very effective at large scale, where the use of a single, static topology to implement various analytics characteristics has substantial negative effects on performance/cost, as shown in Section 6. The hybrid approach also makes it easier for the analytics system to scale-up and scale-down, and to support multiple heterogeneous services and data center structures. In addition, it lends itself to scalability of DCG controllers, as shown in Figure 3. At larger scale, a federation of controllers can be used, where each controller manages a region of the DCG. Finally, we note that DCGs and their regions need not correspond to 'physical zones' in the data center. For example, there could be a 1:1 mapping of DCG regions to zones, or there could be multiple DCG regions within a zone, or a DCG region could span multiple zones if zones are small or if large applications run across all of them.

Figure 3 depicts a typical initial deployment of Monalytics, where there are four DCG regions assigned onto 4 racks, respectively. Each region has a *leader* talking to leaders from other regions. This is a hybrid DCG because each region can deploy a different topology, and the topology between leaders can be arbitrary, as well. There is one DCG controller for each rack; they have access to the M-Brokers in their own rack; and they cooperate with each other to jointly manage DCGs. Our current Monalytics prototype implements one controller for a single region; the development of federation support for controllers is in progress.

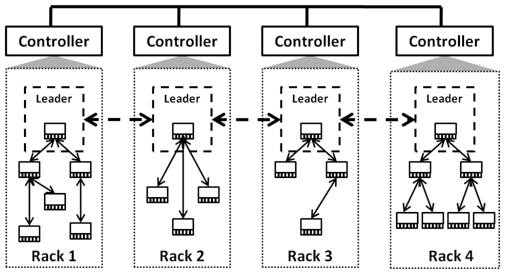


Figure 3: A typical Monalytics deployment on 4 racks

5. MODELING DCG TOPOLOGIES

This section presents a systematic modeling approach to understand Monalytics at scale. These models provide rational estimations and compare the various topologies that Monalytics can create in large scale (from 1000 to 1 million nodes). They are based on real world parameters (network bandwidth, latency, number of nodes, etc.) seen in commercial data centers, and evaluated in realistic configurations, e.g., scale, region size, run times of functions.

Models serve as a sound foundation for the elements of flexibility part of the DCG design. Further, the results obtained from their use reveal new insights on combining monitoring and analytics, including validation of Monalytics' autonomic features like the PRN methods described earlier.

5.1 Traditional Topologies and Hybrid DCGs

Topologies of previous monitoring/aggregation works can be generalized into three types: *centralized, hierarchical tree and binomial swap forest*. In a centralized topology, monitoring data is collected on each node but sent to a central node for analysis. Most of the analysis systems and small to moderate monitoring systems use this approach. Hierarchical trees [40, 33] are widely used in monitoring and aggregation systems, where nodes are organized into a balanced tree (or balanced forest where each tree is according to a different set of attributes), usually with some moderate fanout factor, e.g., 16 [33]. BSF is proposed in [13]. Nodes exchange monitoring data with each other in order and the last two swapping nodes yield the global aggregate data.

Monalytics is capable of creating any of those traditional topologies and in addition, the hybrid DCGs that have significantly higher cost effectiveness in large scale systems, compared to traditional topologies.

5.2 Assumptions

We model the TTI of each topology and the associated management cost. For generality, TTI is defined as the latency between when one monitoring sample (indicating event of interests) is collected on each node and when the analysis on all of those monitoring samples has completed.

Table 2: Parameters

Parameter	Notation	Example value
Number of Nodes	N	$100 - 10^6$[17]
Network Latency	l	0.25ms[17]
Bisection Bandwidth	B	1 Gbps[7]
Bandwidth Budget (in %)	b	0.1% - 1%
Size of Monitoring Data per Node	s	100KBytes
Number of Metrics	n	1890
Processing Time per Metric	a	$3.5 * 10^{-8}$seconds
Capital Cost per Server	c	\$1000[3]
Fraction of Mgmt Cost Per Server	α	1/16[3]
Region Size	N_r	1000

Management cost is modeled as the *capital cost for management hardware and associated software*, i.e., the dollar amount for purchasing management hardware/software resources. We study topologies with dedicated mode and collocated mode in large scale. To the best of our knowledge, our work is the first to model capital cost for management infrastructures in large scale data centers.

Models are based on real word parameters in data centers, as listed in Table 2. The bandwidth resource for each M-Broker in collocated mode is estimated as the product of the bisection bandwidth B and the bandwidth budget b. This estimate reflects the common fact that in many modern data centers, the applications and the monitoring overlay share the same network. It extends on previous models [13] that assume full bandwidth to be available to content aggregation overlays. Further, since monitoring is continuous, which means that it continuously uses the network resource, its bandwidth consumption should be a small fraction of the total bandwidth available in order to confine its interference with applications. Similarly, it is intuitive that on each node, a small portion of resource is used for monitoring/analysis, as captured by a fraction α of its capital cost.

We assume DCs are reporting one monitoring sample at a time. Each sample has a size s due to its use of some number of metrics n. We estimate an intermediate/aggregation result has size s and n metrics as well [2]. We also assume there is support for buffering of raw and aggregated monitoring data throughout the data center, and that there is enough bandwidth provisioned for monitoring for both the dedicated and collocated strategies. M-Brokers failures and associated costs are beyond the scope of this paper.

5.3 Traditional Topologies

Centralized: In a centralized topology, monitoring data collected from each node is sent to a centralized server for analysis. The TTI, noted $T_C(N)$, consists of data delivery time and data processing time. When the aggregate bandwidth of the nodes is smaller than the maximum bandwidth of the central server, the data delivery time is $\frac{s}{B*b} + l$, otherwise it becomes $\frac{s*N}{B} + l$ (central node has full bandwidth because it is dedicated). The processing time is formulated as a function *F(n, M)*, where n is the number of metrics to monitor, and M is the number of instances of each metric. In our analytical models, we assume each node collects n metrics, and uses one instance per metric for analysis, for generality purpose. We can set M to other value representing a different number of instances for analysis, and this will not affect the comparison results shown in Section 6. Therefore, the TTI for centralized DCG is:

$$T_C(N) = \frac{s}{B*b} + l + F(n,N), if B > B*b*N$$
$$or = \frac{s*N}{B} + l + F(n,N), if B \le B*b*N$$

The management cost for the centralized topology has two parts. The first part is the dedicated central server, with cost of C, and the second part is the fraction of management cost on each node. According to empirical experience [3], there is usually one additional dedicated management server for every 1000 nodes added to the data center. Therefore, the cost measurement $C_C(N)$ is:

$$C_C(N) = \lceil \frac{N}{1000} \rceil * c + N * c * \alpha$$

Hierarchical Tree (HT): In a hierarchical tree, internal nodes have similar fanout of at most d, and the leaf nodes have depth of at most $\lceil \log_d N \rceil$ for a system of N nodes. Starting from the leaf level, the nodes at each level can be divided into groups with at most d members. A group can be treated as a centralized topology where children report to their parent. The groups at the same level process data in parallel. Hence, the processing time is $\frac{d*s}{B} + l + F(n,d)$. For the top single group level with the root as the central server, the processing time is $\frac{\lceil \frac{N}{d^{\lceil \log_d N \rceil - 1}} \rceil * s}{B} + l + F(n, \lceil \frac{N}{d^{\lceil \log_d N \rceil - 1}} \rceil)$. A parent analyzes its children's data and sends the results to the next parent at the higher level. This data flow between levels is sequential. Hence, the TTI in dedicated mode is:

$$T_{HT}(N) = (\lceil \log_d N \rceil - 1) * (\frac{d*s}{B} + l + F(n,d)) + \frac{\lceil \frac{N}{d^{\lceil \log_d N \rceil - 1}} \rceil * s}{B} + l + F(n, \lceil \frac{N}{d^{\lceil \log_d N \rceil - 1}} \rceil)$$

If HT is collocated with the monitored system, the bandwidth resource for each node for aggregation is limited. Therefore, the TTI $\bar{T}_{HT}(N)$ is:

$$\bar{T}_{HT}(N) = (\lceil \log_d N \rceil - 1) * (\frac{d*s}{B*b} + l + F(n,d)) + \frac{\lceil \frac{N}{d^{\lceil \log_d N \rceil - 1}} \rceil * s}{B*b} + l + F(n, \lceil \frac{N}{d^{\lceil \log_d N \rceil - 1}} \rceil)$$

The management cost for dedicated mode $C_{HT}(N)$ is the total cost of its internal nodes, of the root, and the inherent cost on each node. The number of parents with N leaves is $\sum_{i=1}^{\lceil \log_d N \rceil} \lceil \frac{N}{d^i} \rceil$:

$$C_{HT}(N) = \sum_{i=1}^{\lceil \log_d N \rceil} \lceil \frac{N}{d^i} \rceil * c + N * c * \alpha$$

In collocated mode, the only cost for management is the fractional management cost on each node:

$$\bar{C}_{HT}(N) = N * c * \alpha$$

Binomial Swap Forest (BSF): In a Binomial Swap Forest (BSF) [13] topology, each node computes an intermediate result by repeatedly swapping (exchanging) data with other nodes. Two nodes swap data by sending to each other the intermediate results they have so far, letting each to compute the new results of both nodes' data. The swaps are organized so that a node only swaps with one other node at a time, and each swap roughly doubles the number of nodes whose

[2]In extreme cases, the intermediate result can be much larger than s, without aggregation, or much smaller, with higher 'compression' effect. We believe it is reasonable to pick a value in the middle as an estimation.

data are processed a node's intermediate result, so that the nodes will compute the global result in roughly log(N) swaps. The sequence of swaps performed by a particular node form a binomial tree with that node at the root. BSF runs in collocated mode and its TTI and cost are:

$$T_{BSF}(N) = \lceil \log_2 N \rceil * (\frac{s}{B*b} + l + F(n,2))$$
$$C_{BSF}(N) = N * c * \alpha$$

5.4 Hybrid DCGs

In hybrid DCGs, any of the traditional topologies described above (centralized, hierarchy, BSF) can be used locally within regions, and also to interconnect region leaders. The resulting graph is thus a hybrid of multiple traditional topologies, same or different, interconnected together. For a hybrid DCG that consists of m *regions* which process their local monitoring data in parallel, the time for leaders to receive the aggregated data would be $\max_{i=1}^{m} T_{INTRA}(R_i)$, where $T_{INTRA}(R_i)$ is the TTI for region i, and R_i is the number of nodes in region i. The region leaders then cooperate to perform the next level processing in a time of $T_{INTER}(m)$. Therefore, the TTI and capital cost of a hybrid DCG are formulated as follows:

$$T_{HYBRID}(N) = \max_{i=1}^{m} T_{INTRA}(R_i) + T_{INTER}(m)$$
$$C_{HYBRID}(N) = \sum_{i=1}^{m} C_{INTRA}(R_i) + C_{INTER}(m)$$
$$(N = \sum_{i=1}^{m} R_i)$$

In practice, there could be numerous topology combinations for a hybrid DCG. For purposes of modeling and evaluation, we pick three representative ones: (1) *Centralized-BSF Monalytics* (CB) uses a centralized topology within each region and a BSF topology inter-region; (2) *Centralized-Hierarchy-BSF Monalytics* (CHB) has centralized or hierarchical topologies within regions and a BSF topology to connect these regions; and (3) *BSF-BSF Monalytics* (BB) uses a BSF topology both intra- and inter-region, and is also 'hybrid' in the sense that the sizes of the intra-region BSF and the inter-region BSF topologies are different. The formulation for the TTI and capital cost for these examples can be obtained by substituting appropriate formulations for centralized, hierarchical, and BSF topologies in the $T_{HYBRID}(N)$ and $C_{HYBRID}(N)$ equations above. For example, assuming all regions have same size N_r for simplicity, the formulations for TTI and capital cost for the *Centralized-BSF* hybrid DCG would be:

$$T_{CB}(N) = T_C(N_r) + T_{BSF}(\lceil \frac{N}{N_r} \rceil)$$
$$C_{CB}(N) = \lceil \frac{N}{N_r} \rceil * C_C(N_r) + C_{BSF}(\lceil \frac{N}{N_r} \rceil)$$

6. EXPERIMENTS AND EVALUATIONS

We have implemented a prototype of Monalytics in C/C++ using the EVPath library [18]. Experimental evaluations at smaller scale use a virtualized environment with 36 virtual machines on 12 physical hosts, complemented by a set of large-scale analytical evaluations to up to 1 million nodes. The results support three main conclusions. First, Monalytics' PRN deployment feature results in substantial TTI reduction compared to traditional approaches. Second, Monalytics has significantly lower cost and interferes less with applications, compared to static systems watching everything all the time. Third, the flexible, hybrid DCG design provides promising advantages in terms of performance and cost, compared to static, single topology solutions (up to 92% TTI reduction and 86% lower cost).

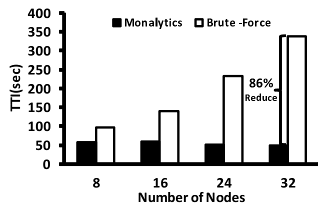
6.1 Experimental Evaluations

Setup: Our testbed consists of 12 blade servers, each hosting 3 VMs. We realize the two use cases described in Section 3, using our Monalytics prototype to monitor/analyze two application benchmarks: (1) RUBiS[14] representing Internet services and (2) Hadoop[1] representing MapReduce applications. We use the 32 out of total 36 VMs as the monitored nodes running RUBiS and Hadoop instances. The remaining 4 VMs are used to emulate user requests to RUBiS and for PRN deployments of new M-Brokers.

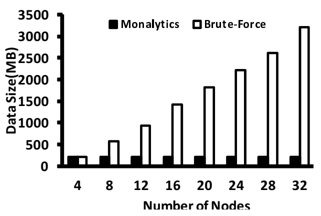
The initial DCG is a hierarchical tree with max fanout of 8. The 32 VMs are leaves, each having one DC and one M-Broker deployed. Some of them are also reused as internal M-Brokers as 1 root and 4 parents. Lightweight anomaly detection [38, 39] and on-line clustering [9] approaches are running on the hierarchy continuously and throughout the experiment. As described in Section 3, the DCG controller creates new centralized DCGs to run deeper analysis functions, causal path inference or Top-k, on demand. Causal path inference needs to gather and merge a considerable volume of traces collected from 'suspicious' VMs, so the transmission of monitoring data has an important effect both on TTI and on application interference, which we will discuss next. The *Top-K* approach analyzes IP packets locally on each candidate cohort member, so the network overhead is low. However, since it parses every input and output IP packet, its CPU consumption is high, which again, can significantly interfere with the applications, especially when running it on every node all the time.

Experiments measure the TTI, monitoring data volume, and interference with applications. We compared our PRN solution with brute-force solutions that 'turn on' analytics functions all the time or on every node. To be more specific, the brute-force ZIA approach collects network traffic trace from all the nodes involved because it does not have the 'zoom-in' capability. By the same token, brute-force VMC runs *Top-K* functions all the time on a VM ensemble. results show that the PRN and in situ approaches result in up to 86% TTI reduction and considerably less interference with application performance. The data size needed for Monalytics is also up to 95% smaller than that of brute-force.

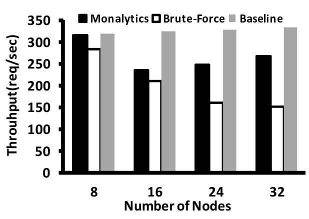
Results: Zoom In Analysis In the ZIA use case, the data center hosting multiple Internet services is monitored, and one of those services has performance problems that requires further analysis via casual path inference. We emulate those problems by injecting anomalies described in [38]. In the experiment, each Internet service is a RUBiS instance embedded in 4 VMs running 1 Apache web server, 2 Tomcat application servers, and 1 MySQL database server, respectively. We vary the system scale from 8 VMs (2 services) to 32 VMs (8 services). Monalytics deploys a DCG on the 4 VMs running the problematic service, on-demand and when needed, while the brute force approach triggers data collection and transmission on all of the VMs. The TTI of Monalytics, then, is significantly shorter than that of the brute force approach, as shown in Figure 4a. As the scale



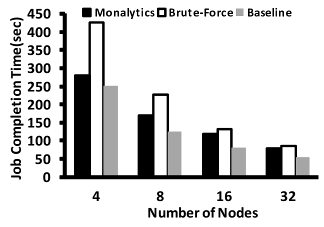
(a) ZIA:TTI



(b) ZIA:Data Size



(c) ZIA:Throughput



(d) VMC:Completion Time

Figure 4: ZIA Results (a), (b), (c) and VMC Result (d)

increases, the TTI gap grows rapidly, indicating even better performance of Monalytics at larger scales, with a 86% TTI reduction on 32 VMs. In addition, the amount of data collected, transmitted, and processed remains low even as the system scales, with the consequent benefits shown in Figure 4b, in comparison to the brute force approach in which the data size increases with increased system scale. Results showing the degree of interference with application performance are encouraging, as well. Here, we measure the average throughput of all RUBiS services as an indication of performance. In Figure 4c, the baseline bar is the average throughput without monitoring/analysis deployed. The Monalytics and brute force bars represent the average throughput for RUBiS with continuous monitoring/analysis. We can see that as the scale increases, the brute force approach results in high interference with application, severely reducing their throughput. This is because the brute force approach consumes substantial network bandwidth to transmit the larger volumes of monitoring data collected, which in turn slows down the performance of the applications sharing the same network. The Monalytics approach has overall lower interference than the brute force approach (up to 44% higher throughput), and the effect of interference decreases as system scales (due to the increased availability of total network bandwidth).

Results: Virtual Machine Clustering In the VMC use case, we run a distributed MapReduce program that uses a BBP-type method to compute the exact binary digits of π [1]. It is run on an infrastructure that scales from 4 VMs to 32 VMs. Figure 4d shows the job completion times when the total workload of the application is fixed, i.e., the workload on each VM is reduced as the system scales. Figure 5a depicts the job completion times when the workload on each slave VM is fixed. In both scenarios, Monalytics incurs much less interferences than the brute force solution, because the former turns on the CPU consuming *Top-K* functions only when needed, whereas the latter runs them all the time, thereby unnecessarily stealing CPU cycles from the Hadoop application. When total workload is fixed, the completion time deceases as the system scales due to the parallel executions on more slaves. The baseline bars in Figure 4d reflect this trend. Accordingly, the effects on completion time decrease, because the application's workload can be finished in a few CPU time slots without interruption. When the per slave workload is fixed, the baseline does not change much because slaves run in parallel, and the job completion time is largely determined by the running time of a slave. Because a slave needs to run a relatively longer time, the brute force approach drags down its performance by running the Top-K algorithm throughout the Hadoop execution. Monalytics induces much less interference on job completion time (an average 12% increase) than brute force (85% increase).

Table 3: $F(n, M)$ Functions Representing Two Types of Analytics, a is the processing time of one metric, and $n * M$ is the total number of metrics

Types of Analytics	$F(n, M)$
Linear-Time Approach	$a * n * M$
Quadratic-Time Approach	$a * n^2 * M^2$

Table 4: Abbreviations used for the Monalytics topologies

Centralized	Centralized
Hierarchical Tree (Collocated)	HT
Hierarchical Tree (Dedicated)	HT-Dedicated
Binomial Swap Forest	BSF
Centralized-BSF	CB
BSF-BSF	BB
Centralized-Hierarchy-BSF	CHB

6.2 Analytical Evaluations at Large Scale

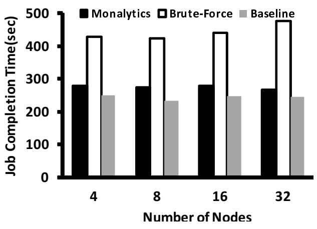
In this section, we evaluate the Monalytics DCG topologies at scale using the models described in Section 5.

Parameters and Estimations: The parameter values used are based on real world practices, and are shown in Table 2 (third column). The characteristics of monitoring data were determined using several runs of a microbenchmark consisting of over 15 well known system monitoring tools. This was used to estimate the monitoring data size and number of metrics per node and per data collection interval[3]. The data processing time on each node, represented by *F(n, M)*, can have various values due to the variety of analytics functions possible. Since it is impossible to exhaust all possibilities, we instantiate *F(n, M)* for our evaluations with two straight-forward and representative run time functions, shown in Table 3. For simplicity, the hybrid DCGs in our evaluations are assumed to have the same region sizes.

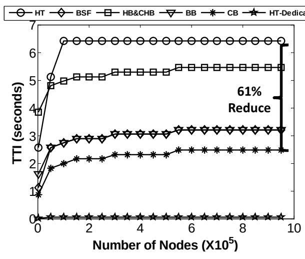
Evaluation Results: We compare the TTI and capital cost of various topologies that can be created by Monalytics (see Table 4 for acronyms). The flexibility offered by hybrid DCGs results in better performance with low cost at scale. In particular, Figures 5b, 5c, and 5d compare the Monalytics' hybrid topologies with traditional HT and BSF topologies when using linear-time and quadratic time analysis functions. As shown in Figure 5b, Monalytics CB provides the second shortest TTI for linear-time analysis.

At the scale of 1 million nodes, the Monalytics CB exhibits a 61% TTI reduction over collocated HT topology. The dedicated HT has the best TTI but also has the high-

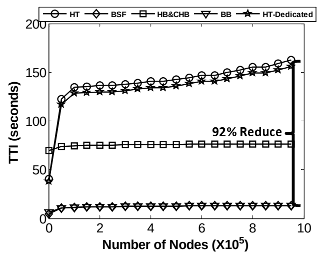
[3]Note that some previous works [13] have used larger estimates of data sizes, dependent on number of application metrics. We did model evaluations with larger data sizes, as well, and found similar results and conclusions.



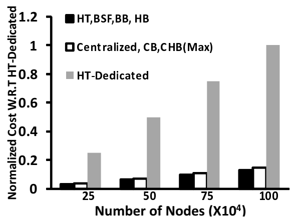
(a) VMC:Completion Time



(b) Model:TTI O(N)

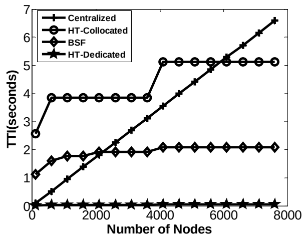


(c) Model:TTI O(N^2)

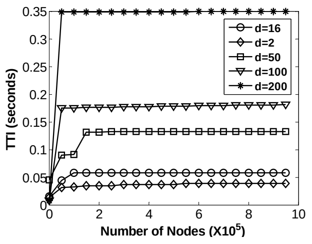


(d) Model:Capital Cost

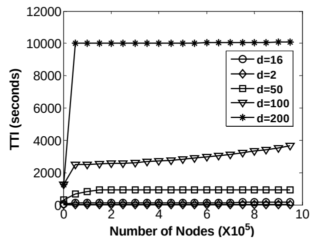
Figure 5: VMC Result (a) and Analytical Model Results (b), (c), (d)



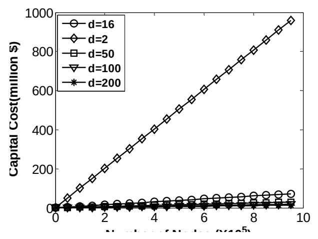
(a) Centralized



(b) HT-Dedicated O(N)



(c) HT-Dedicated O(N^2)



(d) HT-Dedicated Cost

Figure 6: TTI and Cost of Centralized and HT-Dedicated

est capital cost, as depicted in Figure 5d(normalized capital cost with respect to HT-Dedicated cost at scale 1 million). Hybrid DCGs have significantly lower cost, over 85%.

For quadratic-time analysis, Figure 5c shows that Monalytics BB yields the best performance, with a significant 92% smaller TTI than that of the dedicated HT. CB, not shown in Figure 5c, has the highest TTI(over 200 seconds). The computation at each M-Broker dominates TTI for quadratic-time analysis. Since BSF has much less data to compute per node than what HTs have, BSF and Monalytics BB both yield shorter TTI. Conversely, with linear run time, the communication overhead dominates, so the dedicated HT with larger bandwidth out-performs others.

Other hybrid DCGs can perform comparably better than traditional topologies, as well. The Monalytics CBH has better performance than the collocated-HT in both linear and quadratic runtime, and its maximum cost is slightly higher than that of collocated HT, BSF, BB, and HB, but much lower than dedicated-HT, about 86%.

Hybrid DCGs are more effective due to their ability to mix and match the best topologies that meet local analytics requirements and that also perform best at varied local scale levels. In Figure 6a, the completion time of the centralized topology is linear to the scale of data centers, which makes performance prohibitively low at a large scale like 1 million nodes. However, this topology is not always the worst choice because for less than about 2000 nodes, its TTI is less than that of collocated HT and BSF because it has only one level of hierarchy. It also has much lower cost than the dedicated HT, as shown in Figure 5d. Hence, using a centralized topology at appropriate scale may yield good performance. This insight is leveraged to build hybrid CB with encouraging cost-effectiveness in Figure 5b, because for CB, the intra-region centralized topology has smaller TTI than the collocated HT and BSF when the number of nodes is within 2000, and inter-zone processing time is the second smallest.

We also study how topology configurations like fan-out factors affect performance and cost. Figures 6b and 6c reveal that smaller fan-out contributes to smaller TTI. That's because each internal node processes less data with smaller fan-out. Although the height of the tree is larger with smaller fan-outs, the increase has much less effect than that of input data processed by each parent. The improved performance, however, comes with higher cost. As shown in Figure 6d, the dedicated HT with smaller fan-out factors have higher management server costs, because the number of internal nodes increases. The results tell us that configuration may have substantial impact on performance, and the trade-off between performance and cost. Flexibility in DCGs can result in on-demand changes in configurations to meet these different trade-off needs.

Finally, evaluations also show that there is hardly a 'one size fits all' topology for all scale and analysis needs. For example, dedicated HT and Monalytics CB which has best performance in linear runtime analysis are among the worst in quadratic time, which suggests that, instead of using a static, single topology, a dynamic hybrid topology should be applied to meet the changing analytics requirements.

7. RELATED WORK

`Monitoring systems` [31, 21, 8, 35, 20] are designed to monitor the status of large networked systems or large cluster machines. Some of these leverage hierarchical architectures for scalability. Ganglia [31], in particular, uses multicast messages inside a cluster and federations between clusters. While they are widely used and exhibit high performance in reporting states at large scale, their analysis capabilities are limited.

`Aggregation systems` [30, 13, 40, 37, 33] aggregate distributed data with large volumes, and usually provide mechanisms to query the runtime aggregate states. While Monalytics is similar in its ability to perform in-network, distributed processing, there are two major differences. First, Monalytics provides the flexibility to dynamically create, change, and terminate new topologies to meet varying analysis needs. Second, the analysis functions supported by aggregation systems are typically limited to those with the specific computational properties [30] suitable for aggregation, while Monalytics is designed to support a variety of analytics without the restrains.

`Analytics solutions` [5, 6, 10, 11, 15] are promising sophisticated analysis of system behaviors with high accuracy and significantly reduced human intervention. However, they rarely support analytics on-demand and at large scale. In addition, they are often application-specific rather than supporting the general class of utility data center applications targeted by our work.

8. CONCLUSIONS AND FUTURE WORK

We presented the Monalytics software architecture for integrating monitoring and analytics in large scale data centers, with flexibility for supporting a variety of analytics functions. We introduce *pro re nata*(PRN) methods and experimental evaluations are carried out with a Monalytics software prototype implemented in small scale data center running three tier enterprise applications and Hadoop codes. Results clearly show the importance of using PRN, along with the ability of the current Monalytics prototype to support the multiple and sophisticated monitoring/analysis functions required by two realistic use cases. We contribute novel analytical formulations modeling DCG's effects on both the performance and the capital costs of monitoring/analysis, with extensive analytical evaluations in large scale.

Our ongoing work is extending experimental evaluations to scale to multiple thousands of cores and with more complex sets of applications representing modern utility or cloud computing data centers. We are also working on specific extensions of Monalytics software, including federation support for multiple regions and automatically determining appropriate DCGs for specific fault patterns or cost/effectiveness needs. Plus, the resource management/scheduling for concurrent analytics requests in Monalytis is an interesting topic to investigate in the future. Our goal is to routinely run the Monalytics software stack in the data center to better understand actual usage patterns, e.g. finding ways to reduce data center energy use [34].

9. ACKNOWLEDGMENT

This work was funded in part by an HP Innovation Award for "Exascale Systems Management". We thank Dr. Partha Ranganathan, Dr. Dejan Milojicic from HP Labs, and the anonymous reviewers for their insightful comments.

10. REFERENCES

[1] Apache hadoop: http://hadoop.apache.org/.
[2] Hp ilo:http://h18000.www1.hp.com/products /servers/management/remotemgmt.html.
[3] Hp production team: Personal correspondence, sep. 2010.
[4] Ibm systems director:http://www-03.ibm.com/systems/software/director/.
[5] S. Agarwala and etc. E2eprof: Automated end-to-end performance management for enterprise systems. DSN '07.
[6] M. K. Aguilera and etc. Performance debugging for distributed systems of black boxes. SOSP '03.
[7] M. Al-Fares and etc. A scalable, commodity data center network architecture. SIGCOMM '08.
[8] E. Anderson and D. Patterson. Extensible, scalable monitoring for clusters of computers. LISA '97.
[9] R. Apte and etc. Look who's talking: discovering dependencies between virtual machines using cpu utilization. HotCloud'10.
[10] P. Bahl and etc. Towards highly reliable enterprise network services via inference of multi-level dependencies. SIGCOMM '07.
[11] P. Barham and etc. Using magpie for request extraction and workload modelling. OSDI'04.
[12] F. Can. Incremental clustering for dynamic information processing. *ACM Trans. Inf. Syst.*
[13] J. Cappos and etc. San fermin: aggregating large data sets using a binomial swap forest. NSDI'08.
[14] E. Cecchet and etc. Performance comparison of middleware architectures for generating dynamic web content. Middleware '03.
[15] M. Chen and etc. Pinpoint: problem determination in large, dynamic internet services. DSN'02.
[16] I. Cohen and etc. Capturing, indexing, clustering, and retrieving system history. SOSP '05.
[17] J. Dean. Designs, lessons and advice from building large distributed systems. In *LADIS*, 2009.
[18] G. Eisenhauer. PhD thesis.
[19] N. Friedman and etc. Bayesian network classifiers. *Machine Learning*, 1997.
[20] W. Gu and etc. Falcon: On-line monitoring and steering of parallel programs. *Concurrency: Practice and Experience.*
[21] R. C. Harlan. Network management with nagios. *Linux J.*
[22] L. Hu and etc. Net-cohort: Detecting and managing vm ensembles in data center systems. submitted to USENIX ATC'11.
[23] M. Inaba and etc. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract). SCG '94.
[24] E. Kiciman and etc. Ajaxscope: a platform for remotely monitoring the client-side behavior of web 2.0 applications. SOSP '07.
[25] S. Y. Ko and etc. Moara: flexible and scalable group-based querying system. Middleware '08.
[26] A. Kumar and etc. Data streaming algorithms for efficient and accurate estimation of flow size distribution. SIGMETRICS '04/Performance '04.
[27] V. Kumar and etc. A state-space approach to sla based management. NOMS'08.
[28] M. Kutare and etc. Monalytics: online monitoring and analytics for managing large scale data centers. ICAC '10.
[29] J. Leverich and etc. Evaluating impact of manageability features on device performance. CNSM '10.
[30] S. R. Madden and etc. Tag: a tiny aggregation service for ad-hoc sensor networks. OSDI '10.
[31] M. L. Massie and etc. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing.*
[32] F. Murtagh. Complexities of hierarchic clustering algorithms: the state of the art. *Computational Statistics Quarterly*, 1984.
[33] D. Narayanan and etc. Delay aware querying with seaweed. *The VLDB Journal.*
[34] R. Nathuji and etc. Coolit: coordinating facility and it management for efficient datacenters. HotPower'08.
[35] M. Sottile and etc. Supermon: a high-speed cluster monitoring system. Cluster Computing'02.
[36] V. Soundararajan and etc. The impact of management operations on the virtualized datacenter. ISCA '10.
[37] R. Van Renesse and etc. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*
[38] C. Wang and etc. Online detection of utility cloud anomalies using metric distributions. IM'10.
[39] C. Wang and etc. Statistical techniques for online anomaly detection in data centers. IM'11.
[40] P. Yalagandula and etc. A scalable distributed information management system. SIGCOMM '04.