Journal of Big Data
a SpringerOpen Journal

RESEARCH Open Access

v-TerraFly: large scale distributed spatial data visualization with autonomic resource management

Yun Lu[*], Ming Zhao, Lixi Wang and Naphtali Rishe

* Correspondence: yun@cs.fiu.edu
School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA

Abstract

GIS application hosts are becoming more and more complicated. Theses hosts' management is becoming more time consuming and less reliabale decreases with the increase in complexity of GIS applications. The resource management of GIS applications is becoming increasingly important in order to deliver to the user the desired Quality of Service. Map systems often serve dynamic web workloads and involve multiple CPU- and I/O-intensive tiers, which makes it challenging to meet the response time targets of map requests while using the resources efficiently. This paper proposes a virtualized web map service system, v-TerraFly, and its autonomic resource management in order to address this challenge. Virtualization facilitates the deployment of web map services and improves their resource utilization through encapsulation and consolidation. Autonomic resource management allows resources to be automatically provisioned to a map service and its internal tiers on demand. Specifically, this paper proposes new techniques to predict the demand of map workloads online and optimize resource allocations considering both response time and data freshness as the QoS target. The proposed v-TerraFly system is prototyped on TerraFly, a production web map service, and evaluated using real TerraFly workloads. The results show that v-TerraFly can accurately predict the workload demands: 18.91% more accurate; and efficiently allocate resources to meet the QoS target: improves the QoS by 26.19% and saves resource usages by 20.83% compared to traditional peak-load-based resource allocation.

Introduction

With the exponential growth of the World Wide Web, there are more domains open to Geographic Information System (GIS) applications. Internet can provide information to a multitude of users, making GIS available to a wider range of public users than ever before. Web-based map services are the most important application of modern GIS systems. For example, Google Maps has more than 350 million users. There is also a rapidly growing number of geo-enabled applications which consume web map services on traditional computing platforms as well as the emerging mobile devices.

Virtual machines (VM) are powerful platforms for hosting web map service systems. VMs support flexible resource allocation to both meet web map services system demands and share resources with other applications. Virtualization is also enabling technology for the emerging cloud computing paradigm, which further allows highly scalable and cost-effective web map services hosting leveraging its elastic resource availability and pay-as-you-go economic model [1].

However, due to the highly complex and dynamic nature of web map service systems, it is challenging to efficiently host them using virtualized resources. First, typical web map services have to serve dynamically changing workloads, which makes it difficult to host map services on shared resources without compromising performance or wasting resources. Second, a web map service often consists of several tiers which have different intensive resource needs and result in dynamic internal resource contention. Third, for a typical web map service, both response time for requests and the freshness of the returned data are critical factors of the Quality of Service (QoS) required by users.

To address the above challenges, this paper presents v-TerraFly, an autonomic resource management approach for virtualized map service systems, which can automatically optimize the QoS (considering both response time and data freshness) while minimizing the resource cost [2-4]. *First*, v-TerraFly can accurately predict the workload demands of a web map service online based on a novel two-way forecasting algorithm that considers both historical hourly patterns and daily patterns. *Second*, based on the predicted workload, v-TerraFly can automatically estimate the resource demands of its various tiers based on performance profiles created using machine learning techniques. *Third*, v-TerraFly employs a new QoS model that captures the balance between response time and data freshness and uses this model to automatically optimize the resource allocation of a web map service system [5].

This proposed v-TerraFly system is realized on Hyper-V virtual machine environments and evaluated by experiments using real workloads collected from the production TerraFly system. The results show that the proposed two-level workload prediction method is outperforms traditional exponential smoothing prediction by 18.91%, and the system improves the QoS by 26.19% compared to traditional statically node allocation. In the meantime, it saves resource usages by 20.83% compared to traditional peak-load-based resource allocation.

In summary, this paper's main contributions are: (1) created a VM-based map service system, v-TerraFly, which virtualizes all tiers of a typical web map service and supports dynamic resource allocations to the different tiers; (2) proposed a novel autonomic resource management approach for virtualized map services, which automatically allocates resources to different tiers of the service and optimizes the allocations based on the performance and data freshness tradeoff; (3) evaluated v-TerraFly using real workloads collected from production web map service system, which shows substantial improvement on QoS and resource efficiency.

The rest of this paper is organized as follows: Section Background presents the background and motivations; Sections v-TerraFly describes the architecture of v-TerraFly; Section Autonomic Resource Management in v-TerraFly explains the autonomic resource management of v-TerraFly; Section Evaluation presents an experimental evaluation; Section Related work examines related work; and Section Conclusions and future work concludes the paper.

Background

Web map services

As a promising new application of GIS, web map service exhibits its excellence in serving online map requests responsively and delivering geographical information precisely over the Internet [6]. A typical web map service, such as Google maps, Bing maps, and Yahoo maps [7], are usually built upon several major tiers (Figure 1). A *Preprocessor* preloads images and geographic features from raw data repository and splits them into grid format, known as image tiles, to facilitate the *Processor* quickly locating and fetching data. Then a tile *Processor* retrieves and integrates all tiles needed in a customer's query. As its upper tier, a generic map interface accesses this imagery by geo-location, and a client app (or browser) to show the map to end-users.

In this paper, we use TerraFly as a case study of a web map system [1]. TerraFly serves worldwide web map requests, providing users with customized aerial photography, satellite imagery and various overlays, such as street names, roads, restaurants, services and demographic data. Following the typical architecture described above, TerraFly contains two major tiers (Figure 1): the Image *Loader Tier* preprocesses the raw imagery data from repository; the Image *Reader Tier* processes image tiles and retrieves queried images [8]. These tiers are further described in Section v-TerraFly.

Traditionally, web map services are hosted on dedicated physical servers with sufficient hardware resources to satisfy their expected peak workloads in order to provide responsive web services to the users. However, this becomes inefficient for real-world situations where the workloads are intrinsically dynamic in terms of their busty arrival patterns and ever changing unit processing costs [9]. Consequently, peak-load-based resource provision often leads to underutilization of resources for normal state workloads and causes substantial overhead.

Using VMs to host multi-tier web map services can effectively address this limitation because virtualized resources, including CPU, memory, and I/O, are decoupled from their physical infrastructures and can be flexibly allocated to different tiers of the web map system [10]. This approach allows the resource capacity of each tier to elastically grow and shrink to serve its dynamic. In this way, different tiers transparently share the consolidated resources with each other and/or other applications with strong isolation. Such benefits are important to the efficiency of web map service hosting in both typical data centers and emerging cloud systems. On one hand, users need to pay for only the resources their services actually consume. On the other hand, resource providers only

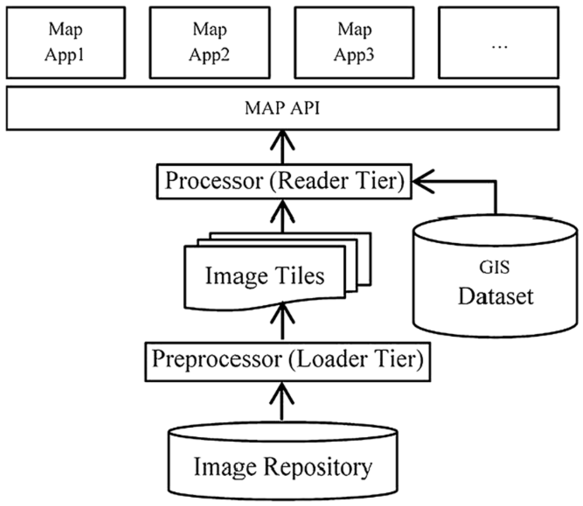

Figure 1 Web-enabled map service.

need to allocate resources as required by the services while saving valuable resources for hosting other applications [11].

Virtualization also offers a new paradigm for web map service deployments. Modern web map services are sophisticated systems, where their installation and configuration require substantial domain knowledge and experience as well as considerable efforts from the administrators. VM-based web map service hosting allows carefully installed software to be distributed as simply as copying the data that represents the VMs. In addition, this approach allows web map services to be quickly replicated and distributed for performance and reliability improvements.

Research design and methodology

v-TerraFly Architecture

To enable the autonomic resource management in TerraFly, we leverage VM techniques to virtualize this multi-tier system, denoted as v-TerraFly. The two critical resource intensive tiers of TerraFly, the image *Loader* and *Reader Tier*s, are deployed on the VMs instead of physical servers.

Figure 2 shows the architecture of this v-TerraFly. The users interact with the application tier which handles most of the business logic and provides advanced application services, such as universal mapping and application specific mapping (e.g. real estate, water management), by sending the mapping queries including position and resolution requirements to the tiers below. Then the image *Reader Tier* is invoked to compute and locate associated map tiles from indexed imagery database according to the requests from the application. To maintain data freshness, the organized imagery database is updated by the *Loader Tier* periodically at the same time. *Loader* contiguously extracts the incoming raw map data from the raw imagery repository, preprocesses and organizes the raw data to destination projection, and then converts the data into the destination file type, and finally updates the data into the organized imagery database.

Due to the structure of the mapping service system, these two tiers of v-TerraFly exhibit distinct resource usage behaviors in the production environment. On one hand, the *Reader Tier* may experience different number of concurrent users during different periods of a day, which results in highly dynamic workloads with varying intensity against the *Loader*. On the other hand, the *Loader* does not have as

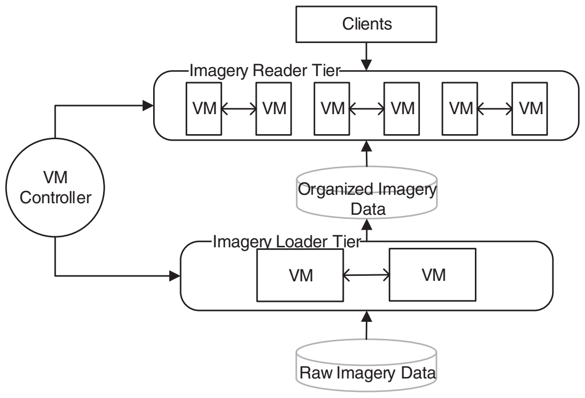

Figure 2 v-TerraFly system.

stringent performance requirement as the *Roader* does but still needs reserved resource to guarantee data freshness. Therefore, it is beneficial to host these two tiers together on virtualized cluster nodes to multiplex the common computing resources so that the total resource capacity can be better utilized among different tiers. For example, more VMs are allocated to the *Reader Tier* during daytime when peak-load of user requests is expected to happen but the loading process is less active; but shifting more VMs to the *Loader* over night to allow data updates accumulated in daytime.

Virtualization in TerraFly also improves the flexibility in terms of the system reliability and scalability. VM is the computing resource in both *Loader* and *Reader Tier*s. With the load balance in both tiers, the work load of each VM is the same; therefore, the VMs in the same tier are considered identical. Since the computing resources can be partitioned through VM nodes, the network bandwidth, which is always a bottleneck in the original system, can now be well balanced among VMs. Furthermore, by pairing every two VMs as complementary *Reader* nodes, it is able to provide more reliable service under unexpected system failure by simply replacing the failed VM with its corresponding back-upped VM [12,13].

Motivating examples

In this section, we demonstrate both the necessity and benefits of resource consolidation in a map service system.

For web map services, the performance of *Reader Tier* and *Loader Tier* are both important. Better *Reader Tier* performance provides shorter page response time; better *Loader Tier* performance provides faster loading of new map data. But both tiers are resource-intensive and they will compete with each other on resource allocation. The goal is to balance the two tiers to achieve best quality of service when we have limited resources.

The workloads of web map service can be highly dynamic over time. Based on the analysis on the web service logs of TerraFly, it is observed that there were millions of web requests received on the *Reader* server over the year of 2012, i.e., more than 450 visits per second on average. However, this workload varies significantly on hourly basis. Figure 3 shows a typical one-day TerraFly workload trace. It shows that the request rate drops to 150 (visits per second) in the morning (around 9:00 am) while rising quickly up to 900 (visits per second) in the afternoon. It would be more efficient to turn off some *Reader* VMs at times.

Assuming the variation in workload which follows such a time-related pattern is predictable, by virtualizing the *Reader Tier* of TerraFly we can easily save resources when the workload intensity is low by simply turning off some *Reader* VMs, and as the workload intensity increases, we can bring back them online to process the additional requests.

To further quantify the resource savings, we replay this one-day trace using two deployment schemes for the *Reader Tier*: the static scheme deploys the Reader tier on the fixed number of computing nodes throughout the entire experiment (2, 4, 6, 8 and 10 nodes respectively); the dynamic schema only assigns sufficient nodes needed by the workload in every hour. The response time is used as the performance metric and the desired QoS target is set to (0.7 s). Figure 4 compares the average response time in every hour using different schemes. We use *node-hour* as the cost unit in terms of

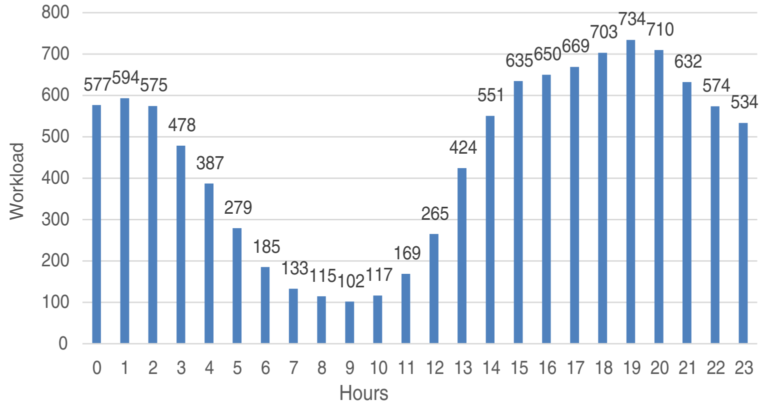

Figure 3 TerraFly system workload hourly distribution.

computing resource. By measuring the number N_i of active nodes used in the ith hour, the total amount of the resource during certain time period T (hours) can be computed as $\sum_i^T N_i$.

Figure 4 compared the measured response times in every hour using different deployments as well as the total resource costs needed in one day. As we can see among the static deployment configurations, only the 10-node configuration can always meet the desired target, however at the cost of the highest total resource amount (240 *node-hour*); others suffer different levels of QoS violation as the workload changes dynamically.

In contrast, the dynamic deployment scheme is able to track the QoS target all the time with only 148 *node-hour,* saving about 23% of the resources compared to the static 8-node configuration which cannot satisfy the QoS target, and saving 38% of the resources compared to the static 10-node configuration which can satisfy the QoS target. Its resource utilization is as efficient as the 6-node configuration but delivers much better performance.

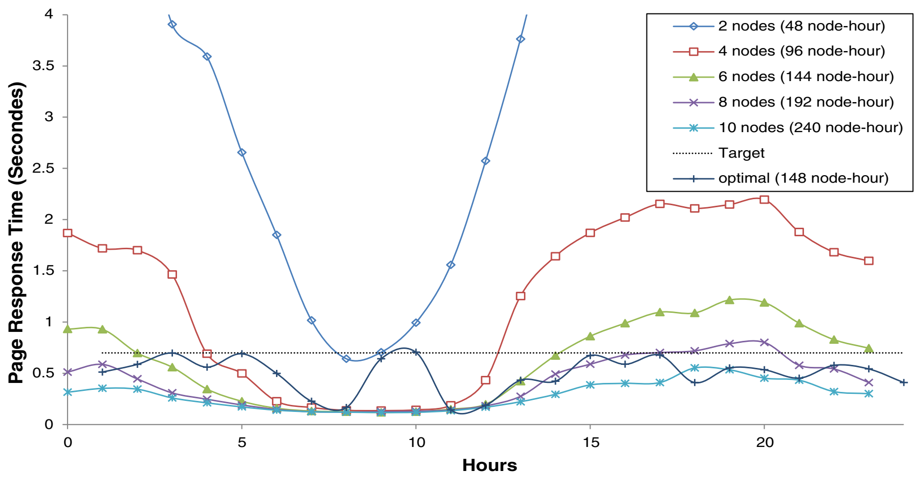

Figure 4 Performance and resource cost comparison using different deployment schemes.

The above example shows strong evidence of the importance of map service virtualization and its online resource management. Currently, our traditional TerraFly system is deployed on the 8 physical *Reader Tier* nodes and 2 *Loader Tier* nodes. It works well for supporting up to 800 concurrent users, and about 6GB fresh data can be load each hour by the 2 *Loader* nodes (Refer to 4.3 Resource Model); but the system scalability is limited due to its fixed physical capacity. It cannot shift resources between *Reader* and *Loader Tier*s even when one tier has idle resource and another has insufficient resources. The inability of shifting resources between tiers results the waste of resources.

However, there are several challenges to dynamic resource management of a virtualized web map service. First, the dynamics in the realistic workload causes the demand of CPU consumption change over time; second, resources also need to be dynamically allocated between the *Reader Tier* to optimize response time and the *Loader Tier* to keep the data fresh.

These challenges can be well addressed by an autonomic VM-based resource management solution which can flexibly partition shared resources and allocate resource on-demand for dynamic workload in order to guarantee performance and improve resource utilization.

Autonomic resource management in v-TerraFly

General approach

Figure 5 illustrates the framework of our proposed autonomic resource management system for v-TerraFly, which consists of three key modules. Please see the notes in Table 1 Parameter description. In this paper, we focus on the resource management for both *Reader* and *Loader Tier*s, since they are the most resource intensive tiers in v-TerraFly.

As a workload executes on the VMs, the *Workload Sensor* monitors the actual workload at current time step, noted as $w(t)$. The *Workload Predictor* then forecasts the future workload $w(t+1)$ for the next time step based on a prediction model. Based on the predicted workload, the *Reader* profile and *Loader* profile which are trained offline are used to estimate their resource demands for time $t+1$, denoted by $rR(t+1)$ and $rL(t+1)$ respectively. The estimated resource demands are then used by the *Resource Allocator* to make the actual allocations by assigning appropriate number of VMs to the *Reader Tier* and *Loader Tier*. Together, these modules form a closed-loop which runs continuously (e.g., every hour), for v-TerraFly's resource control and optimization.

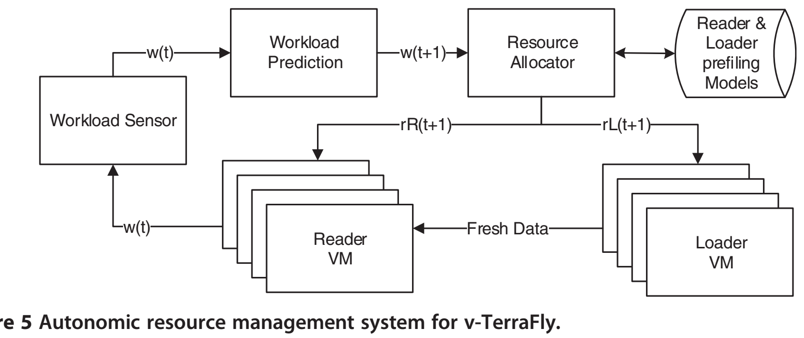

Figure 5 Autonomic resource management system for v-TerraFly.

Table 1 Parameter description

Parameter	Description
$w(t)$	Workload at time t
$rR(t)$	Reader VM CPU resource need at time t
$rL(t)$	Loader VM CPU resource need at time t
w_h^{Des}	Horizontal double exponential smoothing prediction
w_d^{Des}	Vertical double exponential smoothing prediction
w'	Two-level double exponential smoothing

In the rest of this section, we describe the key components of this autonomic system in details.

Workload prediction

In order to accurately and timely predict the workload on v-TerraFly, we propose new forecasting techniques to discover and exploit patterns in user visiting behaviors such as those observed in Figure 3. Specifically, we propose a new two-level time series prediction approach to build a prediction model based on the historical workload measurements, i.e., the request rate observed from the *Reader Tier* of v-TerraFly. Based on such a model, the workload predictor in v-TerraFly is able to estimate the workload intensity for the next time period.

Time series analysis techniques are widely applied in economic data analysis to provide statistical prediction and therefore guide business decisions. A variety of time series prediction methods are available such as the *Moving Averages*, *Linear Regression* and *Exponential Smoothing* [14-16]. In this paper, the TerraFly workload prediction based on the double exponential smoothing (DES) method [17] which is suitable for discrete data sequence with repeated changing patterns.

DES is a smoothing-based forecasting method that can be applied to time series data, a sequence of observations with equally spaced intervals, expressed as $\{Y(0), Y(1),.., Y(t)\}$. Then in DES, the estimate for the $t+1$ time intervals can be computed as:

$$Y^{Des}(t+1) = 2S^{'}(t) – S^{''}(t) + \left(\frac{\alpha}{1–\alpha}\right)\left(S^{'}(t) – S^{''}(t)\right)$$

$$S^{'}(t) = \alpha Y(t) + (1–\alpha)S^{'}(t–1)$$

$$S^{''}(t) = \alpha S^{'}(t) + (1–\alpha)S^{''}(t–1)$$

The equation shows a linear combination of smoothing based statistics associated with a smoothing weight α. The first two components reflect the variation of mean of the overall data while the third tracks the trend of the data. S' is denoted as the singly-smoothed series which smoothed the next measure by assigning a exponentially decreased smoothing weight to the data of the series and computing the weighted average of the observed series. More intuitively, the most recent data is of more importance to the current estimates, i.e., the weight assigned to the data k periods old is $(1 - \alpha)^k$, therefore the closer to 1 of the value of α, less smoothing effect but greater weight to the recent changes. S'' is denoted as the doubly-smoothed series computed by recursively applying the same exponential smoothing operation to the singly-smoothed series *S'* using the same smoothing weight.

In order to perform the time-series-based forecast in v-TerraFly, the workload can be represented as a sequence of intensity measurements that come from a continuing time series at time intervals *T*, denoted as {… *w*(*t*-2 *T*), *w*(*t*-*T*), *w*(*t*)}. More specifically, the workload measurement can be either the average request rate or the number of concurrent client sessions observed in every hour; the time interval *T* can be either one hour or one day (24 hour).

We propose a new two-level double exponential smoothing forecasting model to capture both the daily pattern and hourly pattern of v-TerraFly workload as follows.

$$w^{'}(t+1) = \mu_h w_h^{Des}(t+1) + \mu_d w_d^{Des}(t+1) \tag{1}$$

where w_h^{Des} is the *horizontal* double exponential smoothing prediction based on the hourly pattern in the workload, and w_d^{Des} is the *vertical* double exponential smoothing prediction based on the daily pattern of the workload.

$$w_h^{Des}(t) = 2S^{'}(t-1) - S^{''}(t-1) + \left(\frac{\alpha_h}{1-\alpha_h}\right)\left(S^{'}(t-1) - S^{''}(t-1)\right) \tag{2}$$

$$w_d^{Des}(t) = 2S^{'}(t-24) - S^{''}(t-24) + \left(\frac{\alpha_d}{1-\alpha_d}\right)\left(S^{'}(t-24) - S^{''}(t-24)\right) \tag{3}$$

More specifically, $w_h^{Des}(t)$, called horizontal prediction, is predicted based on { $w(t-3)$, $w(t-2), w(t-1)$} from a hourly series; while $w_d^{Des}(t)$, vertical prediction, is based on the observation series { $w(t-48), w(t-24), w(t)$} that are extracted vertically at the same hours but from continuing days, i.e., a 24-hour vertical time span. The associated μ factors which are set between 0 to 1 are used to balance the importance between three components.

Since each level of DES operation is associated with a smoothing weights, we denote the weights in horizontal and vertical predictions as α_h and α_d respectively. Then the proposed two-level DES model can be considered as a function of α_h and α_d, given observed workload series. Therefore the workload model is trained continuously online as soon as the new measurement is observed by optimizing both α_h and α_d to minimize the weighted sum of squared errors between the prediction and the actual observation. Once the model is updated, it applies to the system immediately for the next prediction.

Performance profiling

Performance profiles relate the workload of the *Reader* and *Loader Tier*s to their resource demands according to the desired performance. Taking the predicted workload $w(t+1)$ as the input, these profiles are used by the *Resource Allocator* to allocate resources to the *Reader* and *Loader Tier*s dynamically in order to achieve the desired QoS.

For the *Reader Tier*, since the workload consists of online web requests, the intensity of the workload is specified by request rate *w*(*t*) as discussed in Section Workload prediction. The relevant performance metric is the average response time *RT*(*t*) of the requests completed during each control period (e.g., one hour). It can be considered as a function of the workload and the number of VM nodes $rR(t)$ allocated to *Reader Tier*. Thus,

$$RT(t) = \varnothing(w(t), rR(t)) \tag{4}$$

The strategy to build this mapping is offline profiling. Given a specific workload w, we map the number of nodes n allocated to the *Reader Tier* to the performance *RT* by iterating over the allocation space and collecting corresponding performance measurements under each allocation candidate. Then we repeat the above step under different workloads by varying the number of the concurrent users in v-TerraFly. Figure 6 illustrates the mapping results by using two to ten *Reader* nodes to serve a workload with 40 to 240 concurrent users. Such a mapping provides the least number of VM nodes needed for a given workload to meet a specific QoS target. For example, if desired response time is set to 0.7 second, then the minimal number of VMs needed is two for a workload with about 40 users and 10 when there are more than 230 concurrent users.

In order to reduce the time required for performance profiling, we collected only a subset of the *Reader* configurations under a subset of the workload intensities, and use linear regression to build the *Reader Tier* entire performance profile. As shown in Figure 7, we got the profile mode and the R square is 0.9131.

We profiled both the CPU and I/O resource usages of the *Reader Tier*, as shown in Figure 7. The results show that both the CPU and I/O demands follow the exact same pattern as the workload varies, which validates the use of identical VM nodes as the resource allocation unit of the *Reader Tier*.

For the *Loader Tier*, since the workload mainly consists of batch jobs which loads raw data into organized repository, the workload intensity is given by the concurrency level, i.e., the average throughput achieved every control period. Allocating more VMs to the *Loader Tier* allows it to obtain higher throughout and finish the loading process sooner. We use offline profiling to create a model for *Loader* that represents the relationship between the throughput (I/O) and the number of VM nodes for the *Loader Tier*. We use different amount of map *Loader* nodes to load a given imagery dataset

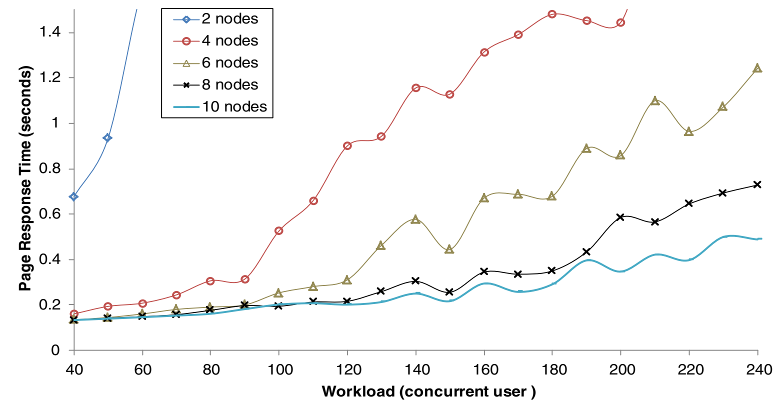

Figure 6 Reader tier profiling (Response Time).

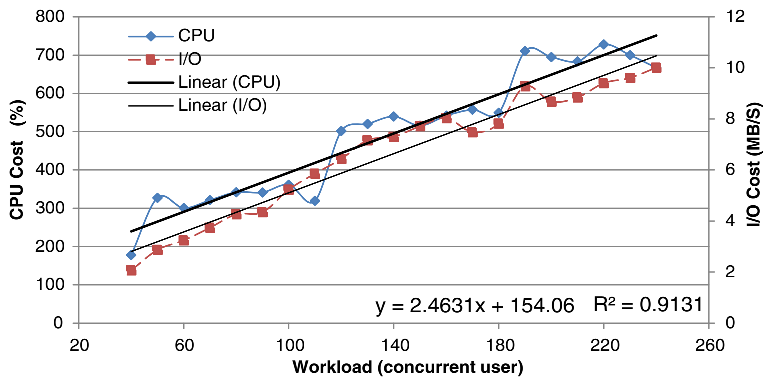

Figure 7 Reader tier profiling (CPU Cost).

and monitor the throughput. We then use linear regression to learn the entire model based on the training data. As shown in Figure 8, the throughput of *Loader Tier* is almost linear with respect to the number of *Loader* VMs. Using linear regression, the R square is only 0.9996.

QoS model

In this section, we propose a novel QoS model to consider both the responsiveness in serving user mapping requests and the quality of returning geographic information. In a virtualized web mapping system, both *Reader* and *Loader* VMs are usually co-hosted in a cluster/data center and compete for the common physical resources, while the former guarantees acceptable response time and the latter keeps the imagery data up to date. Since the performances from both tiers are critical, we need to well balance the importance between them especially when the total resource capacity is constrained. Therefore, a new QoS model is defined to represent the overall system performance at measuring time period t.

$$QoS(t) = r(t) \times f(t) \tag{5}$$

where r(t) and f(t) are the performance metrics for the *Reader* and *Loader Tier* respectively.

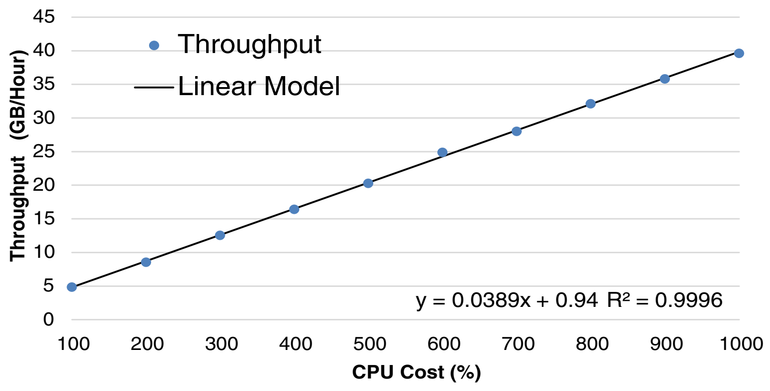

Figure 8 Loader tier CPU usage profiling.

The former QoS component *r*(t) is called the normalized response time, which measures the quality of web mapping services at time *t* and can be calculated as following:

$$r(t) = RT_{ref} \Big/ RT(t) \qquad (6)$$

where RT_{ref} is the desired average response time at *Reader Tier* and $RT(t)$ is the actual performance measurement at time *t*. The higher of the value, the quicker the user requests served at *Reader Tier*.

The latter QoS component *f*(*t*) is called the cumulative data freshness which measures the quality of data-loading process at *Loader Tier* at time *t*. It is calculated recursively based on the previous data freshness value and the current data incremental rate, expressed as following:

$$f(t) = (1-\rho) \times f(t-1) + \Delta D(t)/D_{ref} \qquad (7)$$

where ρ is the decaying factor (predefined in the range of 0 to 1) indicating the data quality loss in terms of freshness during the past time period; D_{ref} is the desired amount of fresh data per time period, $\Delta D(t)$ is the actual amount of data loaded during time period t. Initially phase $t = 0, f(0) = \Delta D(0)/D_{ref}$.

For instance, assuming we need to maintain $D_{ref} = 300GB$ amount of fresh data in repository every control period and the data freshness value at previous time period was $f(t-1) = 0.9$ will be reduced to 0.864 at current time t given a decaying factor of 4.0%. If there is 15GB data loaded during current time period, then the incremental rate is 0.05 and therefore the freshness value $f(t) = 0.864 + 0.05 = 0.914$. Intuitively, we can say the current data quality is 91.4% fresh. Note that it is evident that the data freshness can be adjusted by controlling data incremental rate via resource management at *Loader Tier*.

By maximizing the QoS as computed above, the v-TerraFly resource management system automatically optimize the response time and data freshness simultaneously, which are both important to the map service received by users.

Results and discussion

Setup

This section evaluates the proposed virtual web map service system and its autonomic resource management using the v-TerraFly prototype and real traces collected from the TerraFly production system. As a typical web application, TerraFly usually provides a variety of web services via IIS (Internet Information Services) to serve online web requests. The test bed is set up on two Dell PowerEdge 2970 servers, each with two six-core 2.4GHz AMD Opteron CPUs, 32GB of RAM, and one 1 TB 7.2 RPM SAS disk. Windows Server 2008 and Hyper-V are installed to provide the virtualization environment for v-TerraFly. The resource management system for v-TerraFly is hosted on the hypervisor's management VM. All guest VMs including both *Reader* and *Loader Tier* of TerraFly are installed Windows Server 2008 Data Center as the OS. Each Reader and Loader VM is configured with one core CPU, 2G memory, and 64 GB disk. The resource allocation is done by starting or stopping VMs via Hyper-V PowerShell Script.

Workload prediction

We first evaluate the accuracy of our proposed two-level DES workload prediction algorithm by comparing it to two one-level DES approaches based on hourly pattern only (*Horizontal*) and daily pattern only (*Vertical*) respectively, as well as history average statistics (*History Average*). The evaluation is performed using a real one-month workload trace of the November 2012 extracted from the production TerraFly system's logs. To conduct the experiment more efficiently, the real trace is replayed with a 60-fold speedup, i.e., using one minute in the experiment to simulate one hour in real world. The prediction and updates of the workload models (*smoothing weights* α_h and α_d, refer to Eqs. 2 and 3) are performed every minute to adapt to the dynamics.

Figure 9 compares the online prediction errors of different approaches. Our proposed two-level prediction method delivers significantly better accuracy in predicting the request rate of one month workload. Overall, the 90 percentile average error rate of our two-level method is 10.01% with the lowest standard deviation of 145.3, both much lower than the other three prediction approaches which are 45.67% (*Horizontal*), 28.92% (*Vertical*), and 25.14% (*History Average*) respectively. These results demonstrate that our proposed method can effectively exploit both the hourly pattern and daily pattern in the workload and achieve accurate workload prediction.

Resource management of *Reader Tier*

As discussed in Section Performance profiling, based on the predicted workload, the v-TerraFly resource management system automatically allocates resources to the *Reader* and *Loader Tier*s in order to optimize the QoS. This section evaluates the resource management for the *Reader Tier* alone and demonstrates whether it can achieve the *Reader Tier*'s response time target with the least amount of resources.

In the experiment, a real daily workload trace of October 4th 2012 is replayed against v-TerraFly with a 60-fold speedup. The resource allocator adjusts resources allocation every 1 minute. The QoS target is set to 0.7 s in response time. Based on the profiling, the performance model of *Reader Tier* is $R(t) = 2.4631\ W(t) + 154.06$, (as shown in Figure 7).

Figures 10 and 11 compare the response time and allocations of our dynamic approach to static 6, 8, 9 and 10-node deployment plans respectively. From the results,

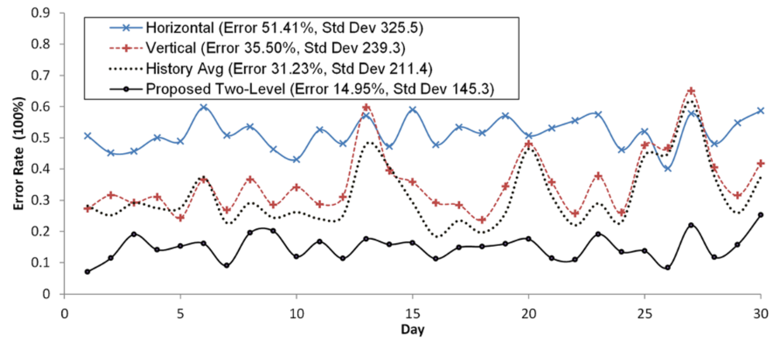

Figure 9 **Error and standard deviation of different workload prediction approaches.**

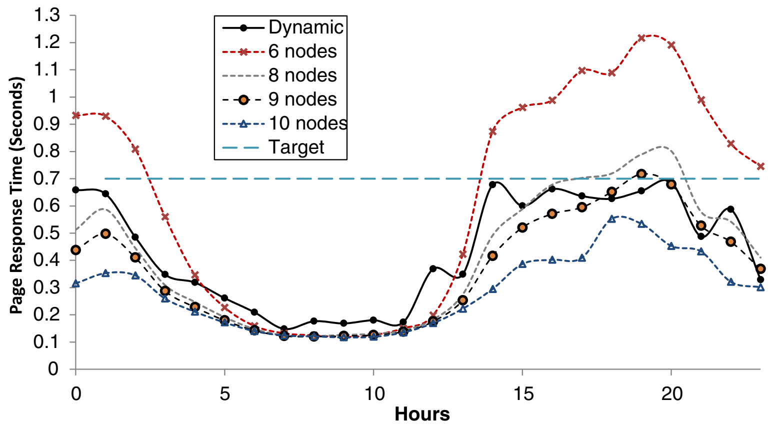

Figure 10 Result: response time.

we can see that the online dynamic approach is able to achieve the response time target all the time throughout the entire experiment. Compared to the 6-node static plan which achieves 0.634 second in average page response time, the online dynamic plan improves the performance by 32.18%, i.e., 0.430 second in average response time, but at the cost of only 5.6% more resource allocation, i.e., 8 additional *node-hour*. There are 13 data points where its response time exceeds 0.7 second in the 6-node static plan, which causes as much as 54.17% QoS violation; in contrast, no QoS violation occurs in the dynamic plan.

Compared to the 8-node static plan, the dynamic plan saves as much as 20.83% of total resources, i.e., 40 *node-hour*. Although the static 8-node plan allocates substantially more with surplus resources, it still causes three QoS violations during the experiment. The 9-node static plan meets the QoS target all the time except the 19th hour, and it costs 29.63% more total resources than the dynamic plan. The 10-node static plan is the only static plan that meets the QoS target all the time, but it costs 36.67% more total resources than the dynamic plan.

Overall, it is evident that the online dynamic deployment plan can efficiently allocate resources to the *Reader Tier* while at the same time meet the response time target by flexibly adjusting its VM assignments in an online manner.

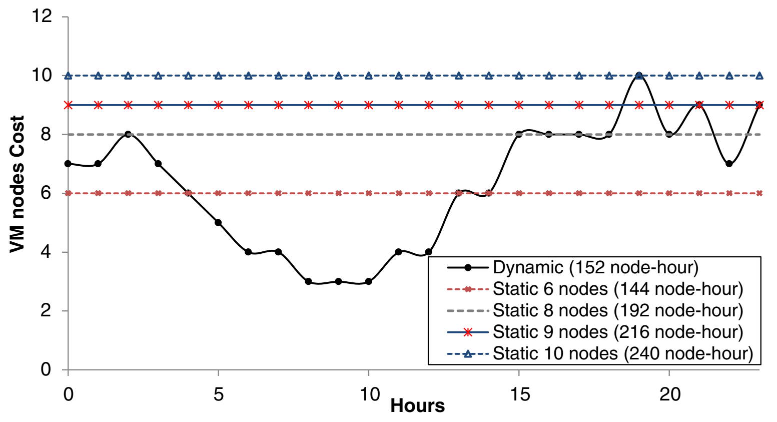

Figure 11 Result: VM nodes cost by hours and total VM nodes cost.

Resource management of both *Reader* and *Loader Tier*s

This section evaluates the proposed autonomic resource management approach for both *Reader* and *Loader Tier*s. Based on the QoS model defined in Section QoS model, the importance of both tiers needs to be balanced in order to optimize an overall QoS value which not only guarantees the responsiveness of map service but also maintains the data freshness of returned maps.

In the experiment, we use the same workload trace described in Section Resource management of *Reader Tier*, and compare our proposed approach to the traditional static deployment plan. The traditional method allocates a fixed number of nodes to *Reader Tier* to satisfy average response time by past experience and gives rest nodes to *Loader Tier*. Specifically, it assigns 7 nodes to the *Read Tier* and 3 nodes to the *Loader Tier* in order to achieve an average response time of 0.9 second and 303.3GB fresh data per day (average $f(t) = 0.809$, refer to Eq. 7).

Figures 12 and 13 compare the performance of the proposed dynamic plan to the traditional static plan in both response time of the *Reader Tier* and throughput of the *Loader Tier*. The static plan achieves an average response time of 0.897 seconds, while the dynamic plan shows slightly better 0.873 seconds. The latter also achieves higher average throughput (194.6 thousands requests per day) than the static one (185.1 thousands requests per day).

Although the performance improvement on the *Reader Tier* is not significant, Figure 14 shows that the proposed dynamic plan achieves much better overall QoS (26.19% improvement). The reason behind this substantial improvement is because the dynamic plan saves resources from the *Reader Tier* and allocates them to the *Loader Tier*, thereby making data loading faster without sacrificing *Reader* performance. Resources are dynamically balanced between these two tiers as the workload changes, where the autonomic resource management allocates only the necessary number of VM nodes to the *Reader Tier* to satisfy current workload, and reserve the rest to *Loader Tier* to load new data.

For example, as showed in Figure 15, from Hour 7 to 10, since the workload on *Reader Tier* is less intense, the dynamic plan allocates more resource to the *Loader Tier* to allow the new data to be loaded as fast as possible. As a result, the dynamic plan loads much more new data (473.3GB Per day) at a varying loading rate than the traditional plan (303.3GB Per day), which loads data at a fixed rate. And the QoS value of

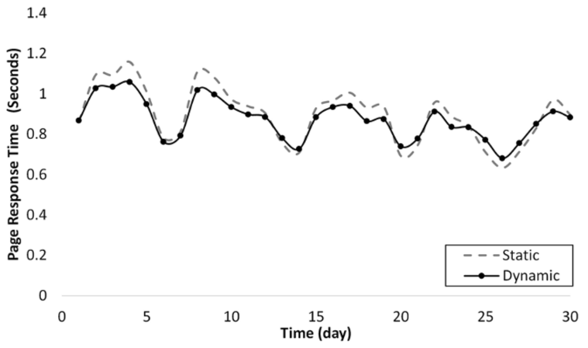

Figure 12 Result: response time improvement.

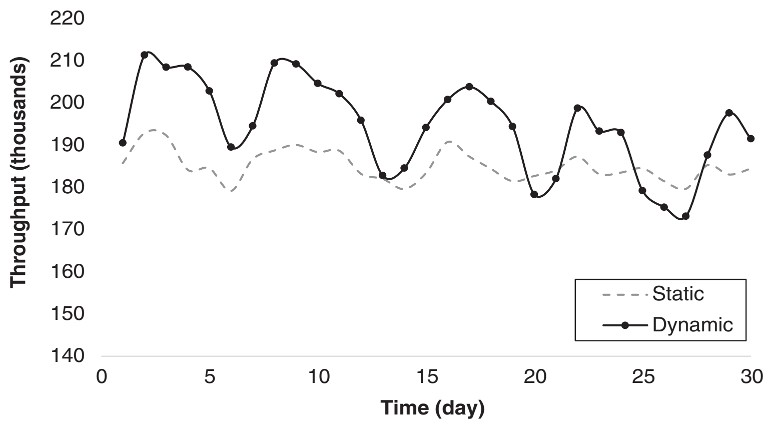

Figure 13 Result: throughput improvement.

the dynamic plan (*Avg QoS = 1.079,* refer to Eq. 5) is *26.20%* higher the traditional plan (*Avg QoS = 0.855,* refer to Eq. 5).

In summary, our proposed autonomic resource management approach is able to automatically optimize the tradeoff between service responsiveness and data freshness by balancing the resource allocations between the *Reader* and *Loader Tiers*.

Related work

In the geospatial discipline, web-based map services can significantly reduce the data volume and required computing resources at the end-user side [18]. To the best of our knowledge, v-TerraFly is the first to study the virtualization of typical web map services and propose QoS-driven resource management for a virtualized web map service through workload forecasting and dynamic resource allocation [19].

Various automatic forecasting algorithms have been studied in the related work [20], including different kinds of exponential smoothing. The work of Brown [16] and Gardner [17] led to the use of exponential smoothing in automatic forecasting for example, like Stellwagen & Goodrich's research at 1999 [15-17]. Hyndman (2002) developed a more general class of methods with a uniform approach to calculate the prediction interval [14,15]. The workload prediction algorithm proposed in this paper

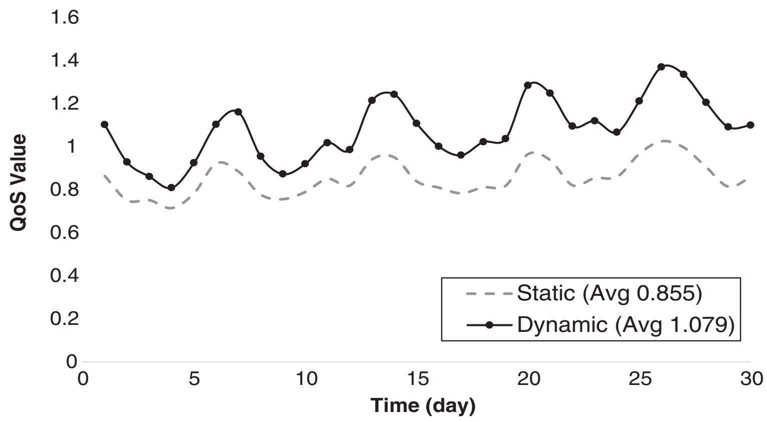

Figure 14 Result: QoS value improvement.

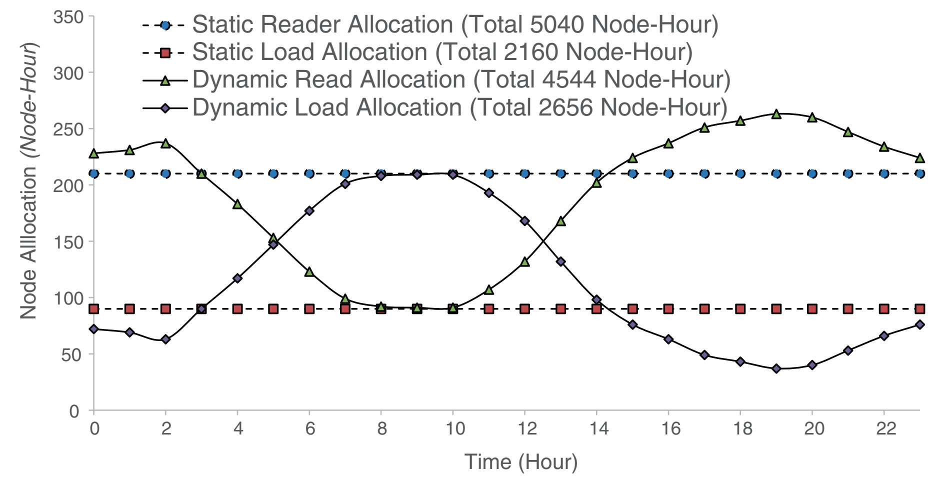

Figure 15 Node allocation per hour.

is based on exponential smoothing, but it is novel in the use of two levels of double exponential smoothing to capture both hourly pattern and daily pattern in the workload, which achieves much higher accuracy than traditional exponential smoothing methods.

In particular, Dinda *et al.* studied prediction-based best-effort real-time service to support distributed, interactive applications in shared computing environments. Two of the examples are an earthquake visualization tool and a GIS map display tool, which were shown to benefit from the service [21]. However, the workload prediction is based on linear prediction which is often not sufficient for real-world dynamic workloads. In this paper, we proposed a two-level exponential smoothing algorithm which shows good prediction accuracy for real TerraFly workloads.

Various types of solutions have been studied in the literature to address the problem of autonomic VM resource management. Different machine learning algorithms have been considered to model VM resource usages [22-25]. Although this article focused on virtualized geo-databases, we believe that our proposed VM resource allocation approach is generally applicable to the virtual resource management for other types of Big Data handling applications. The application-specific part of this approach is geo-database system applications with large numbers of users. Feedback control theory has also been used to adjust VM resource allocations, which are often based on models trained to identify the system and build the controller [26-29]. These various solutions are complementary to this paper's work which focuses on the management of virtualized web map services. Meanwhile, this paper proposes a unique QoS model to capture multiple important objectives and a new method to optimize resource allocation across multiple competing tiers, which has not been studied in the related work and can be applied to manage other multi-tier applications with similar characteristics.

Conclusions and future work

Web map services become increasingly widely used for various commercial and personal uses. Virtualization can greatly facilitate the deployment of web map service systems and substantially improve their resource utilization. To fulfill this potential, the resource management of a virtualized web map system needs to be able to handle the dynamic workloads that the system typically serves and satisfy the often competing demands of the various tiers of the system.

This paper presents a solution, v-TerraFly, to address these challenges. v-TerraFly is created by virtualizing the various tiers of a typical map service system and allowing resources to be dynamically allocated across the tiers. The resource management is done by predicting the workload intensity based on historical data and estimating the resource needs of the map service's *Reader* and *Loader Tier*s based on their performance models. A unique QoS metric is then defined to capture the tradeoff between the service responsiveness and data freshness, and it is used to optimize the resource allocation to the *Reader* and *Loader* VMs.

Experiments based on real TerraFly workload show that our system can accurately predict the workload's resource demands online and automatically allocate the resources accordingly to meet the performance target and save substantial resource cost compared to peak-load-based resource allocation. It can also automatically optimize the tradeoff between responsiveness and data freshness by dynamically balancing the shared resources between the *Reader* and *Loader* VMs.

In our future work, we will improve the scale of v-TerraFly, conducting larger experiments and employing live VM migration as an additional mechanism for optimizing resource management. We will also explore how to apply the principle of v-TerraFly to other applications that have similar dynamic and multi-tier characteristics as a web map service.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
YL carried out the TerraFly deployment pattern studies and v-TerraFly design, participated in the sequence experiment and drafted much of the manuscript. MZ participated the system design, advised and designed most of the experiment and analyzed the experiments results. LW participated in the VM node deployment and helped drafting the manuscript. NR developed the TerraFly technology and major algorithms, designed system layers design and partook in writing the manuscript. All authors read and approved the final manuscript.

Acknowledgement
This material is based in part upon work supported by the National Science Foundation under Grant Nos. MRI CNS-0821345, MRI CNS-1126619, CREST HRD-0833093, I/UCRC IIP-1338922, I/UCRC IIP-0829576, RAPID CNS-1057661, RAPID IIS-1052625, MRI CNS-0959985, AIR IIP-1237818, SBIR IIP-1330943, FRP IIP-1230661, III-Large IIS-1213026, SBIR IIP-1058428, SBIR IIP-1026265, SBIR IIP-1058606, SBIR IIP-1127251, SBIR IIP-1127412, SBIR IIP-1118610, SBIR IIP-1230265, SBIR IIP-1256641, HECURA CCF-0938045, CAREER CNS-1253944. Includes material licensed by TerraFly (http://terrafly.com) and the NSF CAKE Center (http://cake.fiu.edu).

Received: 5 November 2013 Accepted: 20 January 2014
Published: 24 June 2014

References

1. Wang L, Xu J, Zhao M, Tu Y, Fortes JA (2011) Fuzzy modeling based resource management for virtualized database systems. In: Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on IEEE, pp. 32-42.
2. Rishe N, Chen SC, Prabakar N, Weiss MA, Sun W, Selivonenko A, Davis-Chu D (2001) TERRAFLY: A High-Performance Web-based Digital Library System for Spatial Data Access. In: CDE Demo Sessions, pp 17–19
3. Martin P, Elnaffar S, Wasserman T (2006) Workload Models for Autonomic Database Management Systems. ICAS
4. Padala P, Hou K, Shin K, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A (2009) Automated Control of Multiple Virtualized Resources. SIGOPS/EuroSys
5. Kusic D, Kephart JO, Hanson JE, Kandasamy N, Jiang G (2009) Power and performance management of virtualized computing environments via lookahead control. Clust Comput 12(1):1–15. Special Issue on Autonomic Computing
6. Lu Y, Zhang M, Li T, Guang Y, Rishe N (2013) Online spatial data analysis and visualization system. In: Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics, pp 73-79. ACM
7. Craig B (2007) Online Satellite and Aerial Images: Issues and Analysis. North Dakota Law Review 85, pp 547
8. Lu Y, Zhang M, Li T, Liu C, Edrosa E, Rishe N (2013) TerraFly GeoCloud: online spatial data analysis system. In: Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, pp 2457–2460. ACM

9. Lu Y, Zhang M, Witherspoon S, Yesha Y, Rishe N (2013) sksOpen: Efficient Indexing, Querying, and Visualization of Geo-spatial Big Data. In: Proceedings of the 12th International Conference on Machine Learning and Applications (ICMLA'13). IEEE, DOI 10.1109/ICMLA.2013.161 pp 485-490
10. Anderson TE, Peterson LL, Shenker S, Turner JS (2005) Overcoming the internet impasse through visualization. IEEE Comp 38(4):34–41
11. Bennani MN, Menasce DA (2005) Resource allocation for autonomic data centers using analytic performance models. In: Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on IEEE, pp 229–240
12. Baran ME, Wu FF (1989) Network reconfiguration in distribution systems for loss reduction and load balancing. Power Delivery, IEEE Transactions 4(2):1401-1407
13. Wood T, Cherkasova L, Ozonat K, Shenoy P (2008) Profiling and Modeling Resource Usage of Virtualized Applications. Middleware
14. Hyndman R, Koehler AB, Ord JK, Snyder RD (2008) Forecasting with Exponential Smoothing: The State Space Approach. XIII, 362 p
15. Hyndman RJ, Koehler AB, Snyder RD, Grose S (2002) A state space framework for automatic forecasting using exponential smoothing methods. Int J Forecast 18:439–454
16. Brown RG, Meyer RF (1961) The Fundamental Theorem of Exponential Smoothing. Oper Res 9:673–685
17. Gardner ES, Jr (2006) Exponential smoothing: The state of the art—Part II. International Journal of Forecasting 22(4):637-666
18. Lu Y, Zhao M, Zhao G, Wang L, Rishe N (2013) Massive GIS Database System with Autonomic Resource Management. In Proceedings of the 12th International Conference on Machine Learning and Applications (ICMLA'13). IEEE, DOI 10.1109/ICMLA.2013.161 pp 451-456
19. Yue P, Di L, Yang W, Yu G, Zhao P (2007) Semantics-based automatic composition of geospatial Web service chains. Comput Geosci 33(5):649-665
20. Morato D, Aracil J, Diez LA, Izal M, Magana E (2001) On linear prediction of Internet traffic for packet and burst switching networks. In: Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on IEEE, pp 138-143
21. Dinda PA, Lowekamp B, Kallivokas LF, O'hallaron DR (1999) The case for prediction-based best-effort real-time systems. Springer Berlin Heidelberg, pp 309-318
22. Wildstrom J, Stone P, Witchel E (2008) CARVE: A Cognitive Agent for Resource Value Estimation. ICAC
23. Rao J, Bu X, Xu C, Wang L, Yin G (2009) VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration. ICAC
24. Wang L, Xu J, Zhao M, Tu Y, Fortes JAB (2011) Fuzzy Modeling Based Resource Management for Virtualized Database Systems. MASCOTS
25. Xu J, Zhao M, Fortes J (2008) Autonomic Resource Management in Virtualized Data Centers Using Fuzzy-logic-based Control. Cluster Computing
26. Padala P, Hou K, Shin K, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A (2009) Automated Control of Multiple Virtualized Resources. SIGOPS/EuroSys
27. Liu X, Zhu X, Padala P, Wang Z, Singhal S (2007) Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. CDC
28. Lama P, Zhou X (2011) PERFUME: Power and Performance Guarantee with Fuzzy MIMO Control in Virtualized Servers. IWQoS
29. Wang L, Xu J, Zhao M, Fortes J (2011) Adaptive virtual resource management with fuzzy model predictive control. In Proceedings of the 8th ACM international conference on Autonomic computing, pp 191-192. ACM

doi:10.1186/2196-1115-1-4
Cite this article as: Lu *et al.*: v-TerraFly: large scale distributed spatial data visualization with autonomic resource management. *Journal of Big Data* 2014 **1**:4.