SigLM: Signature-Driven Load Management for Cloud Computing Infrastructures

Zhenhuan Gong, Prakash Ramaswamy, Xiaohui Gu, Xiaosong Ma
Department of Computer Science
North Carolina State University
Email:{zgong,pramasw,gu,ma}@ csc.ncsu.edu

Abstract

Cloud computing has emerged as a promising platform that grants users with direct yet shared access to computing resources and services without worrying about the internal complex infrastructure. Unlike traditional *batch* service model, cloud service model adopts a *pay-as-you-go* form, which demands *explicit* and *precise* resource control. In this paper, we present SigLM, a novel *Sig*nature-driven *L*oad *M*anagement system to achieve quality-aware service delivery in shared cloud computing infrastructures. SigLM dynamically captures fine-grained signatures of different application tasks and cloud nodes using time series patterns, and performs precise resource metering and allocation based on the extracted signatures. SigLM employs dynamic time warping algorithm and multi-dimensional time series indexing to achieve efficient signature pattern matching. Our experiments using real load traces collected on the PlanetLab show that SigLM can improve resource provisioning performance by 30-80% compared to existing approaches. SigLM is scalable and efficient, which imposes less than 1% overhead to the system and can perform signature matching within tens of milliseconds.

1 Introduction

Cloud computing has been envisioned as the next-generation Internet service provisioning platform, which allows users to access computing resources and services from the Internet without worrying about the internal complex infrastructure that supports them. Different from traditional Internet service infrastructures such as Grid [9] that support *batch* service model, cloud computing systems (e.g., Amazon's EC2 [1]) support a new *pay-as-you-go* service model. The batch service model allows users to submit their jobs to a batch queue system that is responsible for executing the user's jobs, typically using a certain space partitioning scheduling algorithm [12]. In contrast, cloud systems grant users with *direct* yet *shared* accesses to system resources and charge users for the *exact* resources and services they use (e.g., in terms of service usage time). Thus, unlike batch systems, cloud computing systems demand *explicit* and *precise* resource control. From the cloud service providers' stand point, through more precise monitoring and matching among available resources and application requests, they can improve the overall system utilization and better exploit potential opportunities for hardware and energy saving, both leading to better profit. From the cloud users' stand point, fine-grained resource control allows their computing requests to be better accommodated and enables them to obtain better quality-of-service (QoS) in their jobs' execution.

The key challenges in performing explicit resource control in cloud systems comes from handling the variability and heterogeneity of both application requirements and system resources. Previous work has proposed a range of resource discovery and load management solutions under different distributed computing context. Resource discovery systems (e.g., [17, 4, 7, 20]) are mainly concerned about discovering a subset of candidate nodes satisfying user's resource requirements. Load management systems (e.g., [5]) aim at achieving balanced resource usages among different distributed nodes. However, existing solutions cannot capture the detailed patterns of application workloads and system resources. Thus, the load management systems are forced to use coarse-grained information (e.g., mean, min, max) to either over-provisioning or under-provisioning system resources. Resource under-provisioning affects the QoS perceived by cloud service users while resource over-provisioning hurts the system resource utilization.

In this paper, we present the design and implementation of SigLM, a novel *signature-driven load management system* to achieve QoS-aware service delivery in cloud computing infrastructures. SigLM dynamically captures the precise patterns, namely *signatures*, of application workloads and available system resources using fine-grained time series of different metrics. The system then performs efficient matching between system resources and application workloads based on the dynamically maintained signature patterns. Cloud systems are often used to execute *long-running* computing jobs (e.g, MapReduce [2], stream an-

alytics [10, 13, 11]), which are amenable for the system to observe workload patterns and perform signature-driven load management. SigLM is fully decentralized, which leverages structured peer-to-peer systems [19, 22, 18] to achieve scalable storage and lookup of signature patterns in large-scale cloud systems. To the best of our knowledge, our work makes the first step in applying time series analysis techniques to dynamic load management in large-scale distributed systems.

We need to address a set of new challenges to achieve efficient and scalable signature-driven load management. First, it is more complicated to perform similarity matching between fine-grained time series pattern than between coarse-grained resource specifications. For example, two similar time series may appear very different if one of them is warped or shifted along the time axis. Second, though time series matching pairs resources with workloads more precisely, it is much more time consuming to process a particular load pattern query in a large-scale cloud computing infrastructure that may include tens of thousands of nodes. Third, the signature matching problem is further complicated by the need of multi-dimensional resource requirement, which calls for multiple pattern searches to satisfy simultaneous resource specifications, e.g., regarding CPU and memory. Hence, we need to support multi-dimensional signature pattern matching, which further complicates the signature-driven load management.

To address the challenge, we first develop robust signature pattern matching scheme using the dynamic time warping (DTW) [15] technique. Our scheme can find good matching between two signature patterns even if one of them is shifted or warped in the time dimension, which is particularly important for matching resource and load patterns in distributed computing environments. To achieve efficient signature pattern matching, we develop multi-dimensional signature indexing scheme to achieve fast signature matching in large-scale distributed systems and to support multi-attribute resource-load matching. The signature index provides a fast pre-filtering step to eliminate the majority of dissimilar signatures, which allows the system to execute the DTW algorithm only on those potentially matching signatures. Unlike DTW, which has quadratic complexity, the signature index comparison only takes linear time.

We have implemented a prototype of the SigLM system and deployed it on the Planetlab [16]. We conducted extensive experiments using real-world system resources and application workload traces. Our experiments reveal several interesting findings. First, we observe that our signature-based load management scheme can greatly improve the system resource utilization (i.e., satisfying more requests) than existing coarse-grained information based approaches or statistical resource management approaches. In our experiments, SigLM can improve the system resource utilization by 30% to 80% compared to existing solutions. Given the same set of requests and system resources, SigLM can achieve a much higher resource satisfaction probability and a lower resource violation degree. In addition, the signature indexing scheme can greatly reduce the signature matching time while maintaining the load management efficiency. Our prototype implementation shows that SigLM is feasible for wide-area distributed systems. Using our unoptimized prototype, SigLM can finish signature indexing within several milliseconds and signature matching within tens of milliseconds.

The rest of the paper is organized as follows. Section 2 presents the system model. Section 3 presents the design and algorithms of SigLM system. Section 4 presents the experimental results. Section 5 compares our work with related work. Finally, the paper concludes in Section 6.

2 System Model

We consider a cloud computing system that has N nodes $\{v_1, ...v_N\}$ and runs a set of application tasks $\{t_1, ...t_M\}$. Each node is associated with a set of resource attributes (e.g., available CPU, free memory, disk space) that are denoted by $R = \{r_1, ..., r_k\}$. Each attribute $r_i, 1 \leq i \leq k$ is denoted by a name (e.g., CPU load) and a value (e.g., 10%)[1]. Correspondingly, each application task t_i running in the cloud system is associated with a set of load attributes (e.g., CPU consumption[2], memory consumption, and disk requirements) that are denoted by $L = \{l_1, ..., l_k\}$. Typically, running an application task needs to satisfy multiple resource metrics such as CPU and memory. Thus, SigLM supports multi-dimensional resource-load matching. We use $L \preceq R$ to denote that a task load L is matched by a node resource R, which is defined as follows[3],

$$L \preceq R \Leftrightarrow l_i \leq r_i, 1 \leq i \leq k \tag{1}$$

To achieve fine-grained explicit resource control for cloud systems, we need to characterize both resource patterns of cloud nodes and load patterns of application tasks. On each node, we deploy a monitoring daemon that periodically samples the attribute values of the monitored node and all application tasks running on that node. We use a moving window of time series $Sig_R = \{R_1, ..., R_{|W|}\}$ to represent the current *signature* of the monitored node, where $|W|$ denotes the size of the moving window. Similarly, we use a moving window of time series $Sig_L = \{L_1, ..., L_{|W|}\}$ to represent the current signature of the monitored application task. We define a *satisfaction probability* Pr_s between a node resource signature Sig_R and a task load signature

[1]Unless specified otherwise, we use r_i to represent both name and value of the attribute.

[2]To match a task with different nodes, we assume that the CPU consumption value is properly scaled according to the processor speed difference.

[3]For simplicity, we assume that for all metrics, larger value means more capacity. We can perform some simple transformation on those metrics that do not follow the assumption.

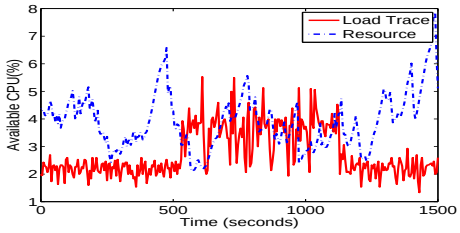

Figure 1. Matching host/task signatures.

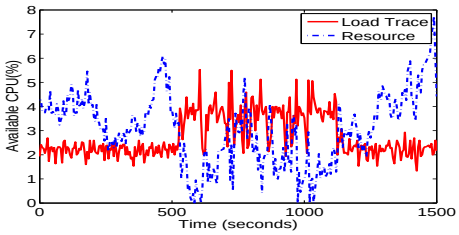

Figure 2. Mismatching host/task signatures.

Sig_L as follows,

$$Pr_s(Sig_R, Sig_L) = \frac{\#\ of\ j,\ L_j \preceq R_j, 1 \leq j \leq |W|}{|W|} \quad (2)$$

The satisfaction probability can better reflect how well the node resources satisfy the task load requirements than coarse-grained information such as mean or current resource values. To illustrate the idea, we use real-word resource and load traces collected on the Planetlab, shown by Figure 1 and Figure 2. If we only consider the mean value, both nodes are considered to satisfy the load requirement. However, the real situation is that host A can accommodate the load most of the time while host B can only satisfy the application resource requirements about 50% of time. By tracking fine-grained resource and load patterns, SigLM can achieve more precise explicit resource control, which is highly desired by the cloud computing infrastructure. For example, given a dynamic application task, we would like to employ a node whose resource patterns match the task load patterns instead of over-allocating resources to meet the task's peak workload requirements. Table 1 summarizes the notations used in this paper.

Compared to traditional coarse-grained load management approach, our signature-driven fine-grained resource control needs to address a set of new challenges: 1) how to identify similar signature patterns without requiring synchronized global clock? 2) how to quickly find matching signature patterns in a large-scale cloud computing infrastructure? and 3) how to support multi-dimensional signature pattern matching? To address the first challenge, we leverage the dynamic time warping (DTW) algorithm to measure the similarity between two time series signature

notation	meaning
N	total number of cloud system nodes
v_i	cloud system node
t_i	application task
r_i	the $i'th$ node resource attribute
$R = [r_1, ..., r_k]$	node resource
R_j	node resource at $j'th$ moving window slot
l_i	the $i'th$ task load attribute
$L = [l_1, ..., l_k]$	task load
L_j	task load at $j'th$ moving window slot
W	moving window of time series
Sig_R	node resource signature in time series
Sig_L	task load signature in time series

Table 1. Notations.

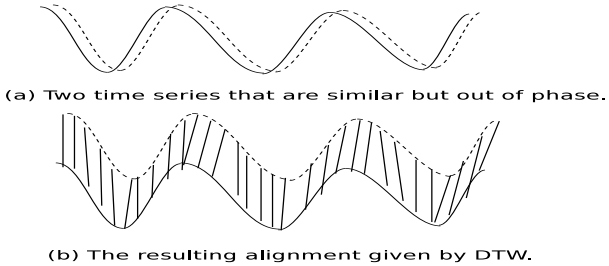

Figure 3. Signature matching using DTW.

patterns that may vary in time or speed. Although DTW is shown to be effective to find an optimal match between two given time series, it has high computation complexity. Thus, to address the second challenge, we employ signature indexing scheme to reduce the candidate set of similar signature patterns on which we need to perform DTW. To address the third challenge, we leverage the R-tree data structure to perform multi-dimensional signature indexing for achieving multi-dimensional signature similarity search. Finally, we leverage P2P distributed hash table (DHT) system [19, 22, 18] to achieve scalable storage and lookup of signature patterns in large-scale cloud systems.

3 System Design

In this section, we present the design details of the SigLM system. We first describe our signature similarity matching algorithm to find optimal cloud node for an application task based on their signature patterns. We then present our multi-dimensional signature indexing scheme that allows SigLM to perform multi-attribute resource matching and speedup signature matching process.

3.1 Signature Pattern Matching

To achieve precise resource control, SigLM captures the signatures of cloud nodes and application workloads using fine-grained time series patterns. Given the load signature

of an application task, the system needs to find a cloud node whose resource signature best matches the load signature[4].

If the signature is represented by coarse-grained information (e.g., mean, min, max), the signature pattern matching can be performed in a straight-forward way. However, if the signature pattern is characterized by time series, the similarity matching becomes much more challenging. To determine the similarity between two time series, we need to define a distance measurement between two time series. A common approach is to use Euclidean distance that is simply the sum of the squared distances from the $n'th$ point in one time series to the $n'th$ point in the other. However, one fundamental challenge is that we cannot assume a synchronized global clock in distributed computing environments. For example, if the signature of a node v_i and the signature of task t_j are identical, but one is shifted slightly along the time axis, a simple distance measurement may consider them to be very different from each other, illustrated by Figure 3 (a). However, v_i can in fact be a perfect match for running t_i since we can align the two time series by starting t_i at the right phase on v_i.

Dynamic time warping (DTW) [15] is a well-known technique for finding the optimal alignment between two time series if one time series may be "warped" or shifted along the time dimension, illustrated by Figure 3 (b). DTW has been widely used in speech recognition, robotics, manufacturing, and medicine. To the best of our knowledge, our work makes the first attempt to apply DTW to fine-grained resource control in distributed systems.

To measure the similarity between a resource time series $Sig_{r_i} = \{r_{i,1}, ..., r_{i,|W|}\}$ and a load time series $Sig_{l_i} = \{l_{i,1}, ..., l_{i,|W|}\}$, we construct a $|W|$-by-$|W|$ matrix where the ($k'th$,$m'th$) element of the matrix denotes the distance $d(r_{i,k}, l_{i,m})$ between the two points $r_{i,k}$ and $l_{i,m}$ (e.g., $d(r_{i,k}, l_{i,m}) = (r_{i,k} - l_{i,m})^2$). A warping path WP is a contiguous set of matrix elements that define a mapping between Sig_{r_i} and Sig_{l_i}. The $q'th$ element of WP is defined as $wp_q = (k, m)_q$. So we have

$$WP = wp_1, ..., wp_q, ...wp_Q, |W| \leq Q < 2|W| - 1 \quad (3)$$

The warping path has to satisfy a set of constraints including boundary conditions, continuity, and monotonicity [14]. There are exponentially many warping path that can satisfy those constraints. The goal of DTW is to find the warping path that minimizes the warping cost.

$$DTW(Sig_{r_i}, Sig_{l_i}) = min\sqrt{\sum_{q=1}^{Q} wp_q} \quad (4)$$

We can use dynamic programming to find the minimum cost warping path. More details about the DTW algorithm can be found in [15]. DTW has time and space complexity of $O(|W|^2)$, where $|W|$ is the number of points included in the time series (i.e., the size of the moving window).

[4]In this paper, we use the best-fit load matching policy to explain our algorithm. Our approach can be readily applied to other load matching policy.

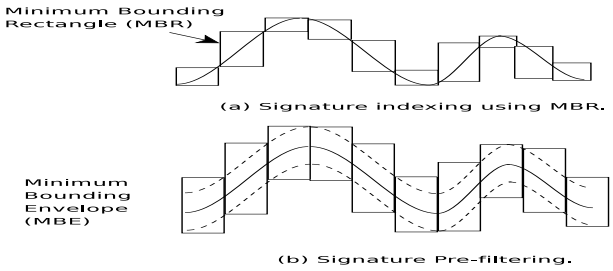

Figure 4. Signature indexing and pre-filtering.

3.2 Multi-Atttribute Signature Indexing

Although DTW provides excellent time series similarity matching performance, it can become bottleneck in our system when we need to perform signature matching between a large number of cloud nodes and application tasks. Moreover, DTW has been used so far for one-dimensional time series. However, in practice, load management needs us to match multiple resource metrics at the same time. To address the problem, we leverage multi-dimensional time series indexing [23] to speedup the signature matching process and support multi-attribute signature matching. The basic idea is to employ a fast pre-filtering step to eliminate the majority of dissimilar signatures and execute costly DTW algorithm only on some potentially matching signatures.

We construct the index for a multi-attribute node resource signature Sig_R as follows, which is illustrated by Figure 4 (a). For clarity, Figure 4 only gives a one-dimensional time series example. First, we split the time series into a sequence of segments based on a pre-defined segment length (e.g., 10 measurement points). We then construct a *Minimum Bounding Rectangles* (MBRs) for each segment. We form the MBR by taking the lowest and highest value within the segment as the lower bound and upper bound of the MBR. Note that if the time series is multi-dimensional, the corresponding MBR is also multi-dimensional. We then store the coordinates of those multi-dimensional MBRs into a R-tree. R-trees are tree data structures that are similar to B-trees, but are used for indexing multi-dimensional information.

Index-based Signature Pre-Filtering. Given a load signature Sig_L, we first construct a *Minimum Bounding Envelop* (MBE) around Sig_L given a pre-defined range of possible matching, illustrated by Figure 4 (b). Suppose we set the matching range as 2%. We then scale-up the time series by 2% to get the upper-bound of the MBE and then scale-down the time series by 2% to get the lower-bound of the MBE. We then split the MBE into a sequence of segments and construct a set of MBRs for all segments. In this case, the MBR is formed by taking the lowest and

highest value in the MBE as the lower bound and upper bound of the MBR, respectively. We then perform pre-filtering by calculating the MBR intersections with those node resource signature MBRs stored in the R-trees. We say that an MBR of the load signature Sig_{l_i} can be matched by an MBR of the resource signature Sig_{r_i} if the lower-bound of the load MBR is lower than the upper-bound of the resource MBR. We define a *pre-filtering qualifying function* that a resource signature Sig_{r_i} is considered as *qualified* if the MBR matching is larger than a certain threshold (e.g., 80% MBRs of the resource signature matches the MBRs of the load signature).

For *multi-attribute load matching*, we can define qualifying functions for different resource attributes (e.g., CPU, memory, disk) separately. We say that a multi-dimensional resource signature Sig_R is qualified for a multi-dimensional load signature if the qualifying functions for all dimensions return positive results. Only on those qualified resource signatures, we execute the DTW algorithm to find the best signature match. Note that MBR-based pre-filtering algorithm is much faster than DTW, which has linear time and space complexity of $O(|W|)$.

3.3 Dynamic Load Management

SigLM provides dynamic runtime load management for executing long-running data-intensive computing jobs in cloud systems. To achieve runtime load management, each cloud node needs to periodically update its multi-attribute resource signatures. SigLM performs dynamic matching between currently running tasks and existing cloud nodes based on the maintained load and resource signatures. For each newly arrived task, the system first instantiates the task on some lightly loaded node to collect the task's load signature. Figure 5 shows the pseudo-code of the major algorithm steps in the SigLM system.

To accommodate large-scale cloud systems, we leverage P2P distributed hash table (DHT) system [19, 22, 18] to achieve scalable storage and lookup of cloud node resource signatures. Each node maintains several sliding windows of recent measurements for a set of resource metrics as its resource signature Sig_R. The node then constructs the multi-dimensional indexing for Sig_R (i.e., R-Tree) using the algorithm described in the previous section. The node periodically pushes its resource signature and its index into the DHT system.

When the system needs to assign a newly arrived task or migrate an existing task, the system generates a load matching request by pushing the task load signature into the DHT system. Upon receiving the matching request for a load signature Sig_L, each DHT node performs pre-filtering by comparing the index of Sig_L with the indexes of node resource signatures stored locally using the algorithm described in the previous section. Each DHT node then sends those matching cloud node resource signatures that pass the pre-fitlering qualifying function back to the cloud node initiating the load matching request. The cloud node aggregates the lists of qualified candidate cloud nodes received from different DHT nodes, and then invokes the DTW algorithm to calculate matching scores between the task load and all qualified nodes. The system then picks the cloud node that has the highest matching score and passes the admission control function to host the task.

```
Input:
V = {v_1, ..., v_n}: nodes in the cloud system
t_i: a task that needs to be placed in the cloud system
W: signature sliding window
f_i: pre-filtering qualifying function for resource type r_i ∈ R
DHT: P2P signature lookup system

UpdateResourceSignature(V, |W|, DHT)
1.  for every signature window |W| do
2.      for each node v_i in V do
3.          for each resource attribute r_k in R do
4.              Construct the resource signature Sig_{r_k}
5.              Construct the index MBRs for Sig_{r_k}
6.              Insert the MBRs into R-trees
7.              Push Sig_{r_k} and its index into DHT

MatchTaskSignature(t_i, V, DHT)
1.  Construct the MBRs for the load signature Sig_L of t_i
2.  Send load matching request to DHT nodes
3.  for each DHT node do
4.      for each cloud node resource signature Sig_R do
4.          flag = TRUE;
5.          for each resource type Sig_{r_i} do
6.              flag = flag ∧ f_i(Sig_{l_i},Sig_{r_i}) /* MBR matching*/
7.          if (flag == TRUE)
8.              insert the cloud node signature into a DTW list
9.      return the DTW list to the initiating node for t_i
4.  merge DTW lists received from all DHT nodes
7.  for every node resource in the DTW List do
8.      Invoke DTW algorithm to get a matching score α
9.  Sort the DTW List based on α
10. for every node v_j in the sorted DTW list do
11.     Invoke admission control func. between t_i and v_j
12.     if admission control func. returns TRUE
13.         Allocate t_i to v_j
14.         Break
```

Figure 5. SigLM load management algorithms.

4 Experimental Evaluation

We have implemented the SigLM system and conducted both extensive trace-driven experiments and prototype evaluation on the PlanetLab [16]. In this section, we first

describe our system implementation and experiment setup. We then present our experiment results.

4.1 Experiment Setup

To perform extensive controlled experiments, we perform trace-driven experiments where SigLM node software is fully implemented but only task load and node resources are emulated. We have collected real application workload and node resources on the PlanetLab to drive the trace-driven experiments. Specifically, we collect a set of resource metrics (e.g., CPU, memory) to indicate the available resources on different PlanetLab nodes. We also collect application load metrics (e.g., CPU load, memory consumption) of those applications running on the PlanetLab nodes. In trace-driven experiments, we set the node resources and task load requirements based on the collected traces. We also conducted prototype evaluation of the SigLM on the PlanetLab by running computation-intensive applications on top of SigLM.

We use DTW and $DTW + Indexing$ to denote the two variations of our approach. The DTW means that SigLM employs the DTW algorithm to perform exhaustive signature matching between each task and each node without any indexing assistance. The $DTW + Indexing$ denotes the SigLM system with the MBR-based signature index. For comparison, we also implemented a set of common alternative load matching algorithms: 1) "$Histogram$" denotes a statistical matching algorithm. We construct histograms for each metric by counting the frequency of metric value falling into different value range (i.e., different bins). Given a load matching request, we calculate matching score between the resource histograms of different nodes and the request histogram by computing a weighted sum of the difference for every bin[5]. We choose the node that has the smallest positive matching score as the best-fit cloud node to execute the request task; 2) "$Mean$" denotes conventional coarse-grained load matching algorithm that uses the metric average value as the resource/load signatures; and 3) "$Random$" denotes a random load matching algorithm that simply allocates a task to a randomly selected nodes. Note that for all algorithms, we periodically invoke the load matching algorithm for every signature window period. For example, if the signature has 100 measurement points and each point is sampled every 10 seconds, then we invoke the load matching algorithm between all running tasks and all system nodes every 1000 seconds. The arrival of application requests to the system follows poisson distribution with a mean-inter arrival time of ten seconds.

We evaluate different load management algorithms under two different usage scenarios. In the first set of experiments, we perform admission control for each load matching request. In our experiments, we consider resource signature values that are not within the 2% of the request signature as violation. If less than 35% of such violation were found, we say such a request can be admitted into the system. The number of requests satisfied is measured with this admission control. Note that those parameters can be configured and do not affect the relative merits of different algorithms, illustrated by Figure 8. In the second set of experiments, we do not perform admission control and instantiate all requests. We measure the *resource satisfaction probability* Pr_s (equation 2) and *resource violation degree* by comparing the required resources with available resources at a particular time t. The violation rate and violation degree are computed by considering the best possible resource signatures according to the algorithm. For all the experiments other than the one described in Table 2, the MBR size is fixed at five samples and signature window size as 500 samples. Each experiment run is repeated ten times and the mean of them is reported. In all experiments, we perform multi-attribute load matching considering both CPU and memory requirements. The criteria for evaluating multi-attribute load matching is described in Section 3.2.

[5]We assign higher weights to the bins representing larger value range since the positive difference at larger value range weighs more than the positive difference at lower range value for resource satisfaction.

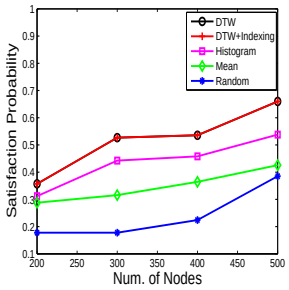

Figure 9. Satisfaction rate under different system sizes.

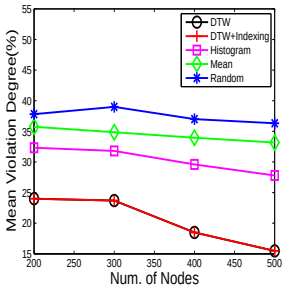

Figure 10. Violation degree under different system sizes.

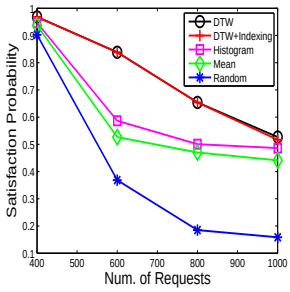

Figure 11. Satisfaction rate under different request loads.

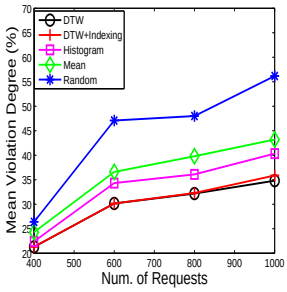

Figure 12. Violation degree under different request loads.

4.2 Results and Analysis

We conduct the first set of experiments to compare the performance of our algorithm with traditional load manage-

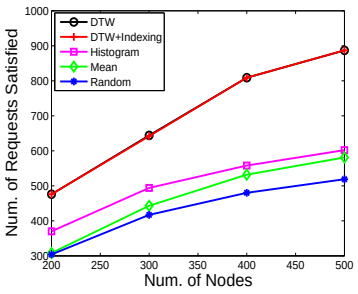

Figure 6. Resource utilization under different system sizes.

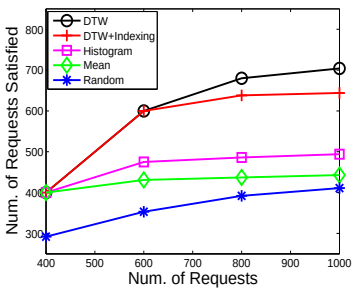

Figure 7. Resource utilization under different request loads.

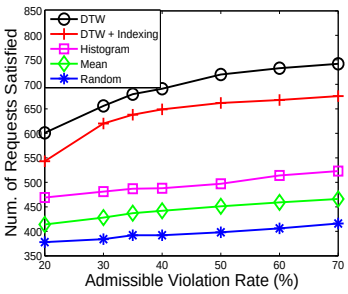

Figure 8. Resource utilization under different admission control threshold.

ment schemes with admission control. We first fix the number of application requests at 1000 and gradually increase the number of nodes from 200 to 500. Figure 6 shows the number of satisfying requests that can be achieved by different algorithms. We observe that both DTW and DTW+Indexing can admit far more requests than previous algorithms. This shows that fine-grained load management scheme is effective for real application workloads. We also observe that the signature indexing did not affect the signature matching performance, which means the pre-filtering step only filters out unmatched signatures. Figure 7 shows the number of satisfying requests that can be achieved by different algorithms under various number of requests. The number of nodes is fixed at 300 and the number of requests is varied from 400 through 1000. Again, we observe that our algorithms can achieve much better resource utilization than traditional approaches. In this set of experiments, we observe that DTW with indexing achieves slightly worse performance than DTW due to the pre-filtering step.

We also conduct sensitivity study to show that the advantage of our approach is not affected by the admission control threshold. In the above experiments, we set the admission control threshold as 35%, that is a request can be admitted into the system if less than 35% of multi-attribute resource violation were found. Figure 8 shows the number of satisfying requests that can be achieved by different algorithms under different admission control threshold. Higher admission control threshold means higher violation rate is allowed by the applications. Thus, more applications will be admitted into the system. We observe that our algorithm consistently admits more requests than previous algorithms given the same set of cloud nodes.

We then repeat the same experiments but remove the admission control. Figure 9 shows the satisfaction probability Pr_s (equation 2) achieved by different algorithms under different system sizes. We still fix the number of requests at 1000 and measure the mean satisfaction probability among all 1000 requests. Different from previous set of experiments, we don't perform any admission control and match each request with the best node based on the time series or mean value matching algorithm. Note that such a matching is performed periodically every signature window $|W|$ for each application task. We also observed that DTW and DTW + Indexing can achieve much satisfaction probability than the other alternatives. Figure 10 shows the resource violation degree measured in the same set of experiment. We also observe that DTW and DTW with indexing can achieve much lower violation degrees than other approaches. For example, for 500 nodes and 1000 requests, DTW has 15.472% violation degree compared to 27.5% for the Histogram algorithm and 33.17% for the Mean algorithm.

We then fix the number of nodes at 300 and increase the number of requests from 400 to 1000. Figure 11 shows the resource satisfaction probability Pr_s compared under different request numbers. We observe that with increasing number of requests, the probability that a load request is satisfied by its host resources decreases. But the extent of decrease is the least for the DTW and DTW with indexing techniques as they always try to allocate a best-fit at a finer granularity. Figure 12 shows the resource violation degree collected in the same set of experiments. We observe that with increasing number of requests, the mean violation degree increases. But the extent of increase is the least for the DTW and DTW with indexing techniques for the same reasons discussed earlier.

We now evaluate the performance of our signature indexing algorithm. The length of MBR decides the filtering effect of the signature indexing. In Table 2, the filtering ratio is shown for different MBR sizes. We observe that with increasing MBR size, the filtering ratio reduces as the granularity level decreases. The reason is that with bigger MBR size, each rectangle covers more area in the time series and thus achieve less filtering power. However, bigger MBR size means fewer MBR comparisons in the pre-filtering step. Thus, we can adjust the MBR size to achieve a proper tradeoff between the filtering ratio and indexing overhead.

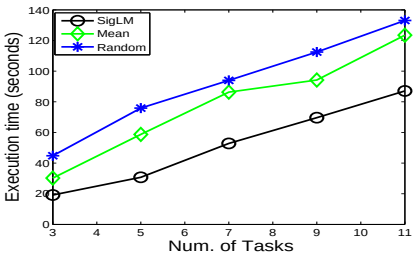

Figure 13. Application execution time comparison on the Planetlab.

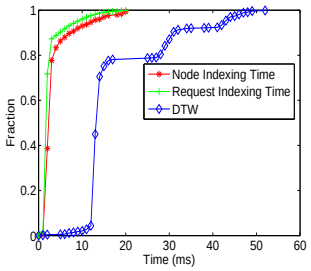

Figure 14. Signature indexing and matching time.

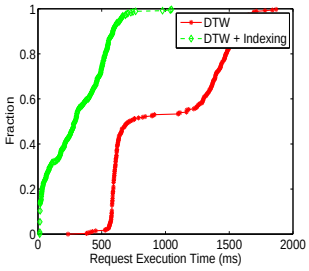

Figure 15. Total request processing time.

As a proof-of-concept, we implemented a simple distributed sorting application on top of the SigLM system. We deploy the SigLM system on about 20 PlanetLab nodes and generate a sequence of sorting application requests. We generated application tasks by invoking the bubble sort executable on a set of numbers from 1 to n (n generated at random for every task). We use different algorithms to map tasks to different PlanetLab nodes and measure the total completion time for all tasks, shown by Figure 13. We observe that SigLM can achieve lower execution time compared to other alternatives. We also measured the overhead of the SigLM system. Figure 14 shows the cumulative distribution function (CDF) of the processing time taken to match a resource signature with a load signature. Additionally, the time spent in indexing the load and resource signatures are also shown. It can be seen that, since indexing is a linear algorithm, most fraction of its time spent is within several milli-seconds. The DTW matching algorithm takes more time, which needs tens of milli-seconds. Figure 15 shows the time taken to process each request by matching the request load signature with the resource signatures of all the nodes in the system. We can see that indexing can significantly reduce the request processing time in our system. Our system also requires more storage to store those signatures. We need 15KB to store a two-attribute signature with a window size of 500 data points. We believe such a storage overhead is very small compared to the storage capacity of modern computers. Moreover, we can leverage DHT to distribute the storage overhead among different distributed nodes.

MBR size	5 samples	20 samples	100 samples
Filtering ratio	68%	63%	52%

Table 2. The impact of MBR size on index filtering ratio.

5 Related Work

Previous research work has proposed different distributed system resource discovery schemes. For example, Gangmatching [17] provides a multi-lateral matchmaking model that employs classified advertisement to describe entity and policy constraints and preferences. SWORD [4] is a wide-area resource discovery system that allows resource requirements to be defined as range of accepted values and stricter range of preferred values. SWORD supports multi-attribute resource discovery by converging multi-dimensional attributes into a single dimension using linear clustering. PIRD is a P2P-based intelligent resource discovery system that weaves multiple attributes into a set of indices using locality sensitive hashing, and then maps the indices to a structured P2P system. Different from the above work, SigLM supports dynamic signature-driven resource discovery and matchmaking, which can achieve better resource utilization in cloud computing systems.

Resource Bundles system [7] provides a histogram-based statistical resource discovery and allocation scheme, which employs clustering to achieve scalability. However, resource bundles system does not support multi-dimensional resource matching and histogram-based approach cannot achieve precise resource control expected by the cloud system. In contrast, SigLM supports time-series signature-based load management and employs R-trees to achieve scalable signature-driven resource matchmaking.

Recent studies have shown that recognizing system patterns is a promising approach to automatic system management [8, 6, 21]. Our work is similar to the above by adopting a signature pattern driven approach. However, to the best of our knowledge, our work makes the first attempt to achieve scalable signature-driven fine-grained load management in the context of cloud computing.

6 Conclusion

In this paper, we have presented SigLM, a new signature-driven load management system for large-scale cloud computing infrastructures. Different from traditional coarse-grained approaches, SigLM can capture detailed patterns of dynamic system resources and application workloads using fine-grained, dynamically updated, time series signatures.

Thus, SigLM can perform more efficient resource provisioning based on dynamically maintained signature patterns. SigLM provides robust signature matching algorithm using dynamic time warping technique and employs multi-attribute signature index to achieve fast signature matching in a large-scale distributed system. To the best of our knowledge, SigLM makes the first attempt to apply time series analysis techniques to achieve more efficient load management in large-scale distributed infrastructures. We have implemented the SigLM system and deployed it on the PlanetLab testbed.

We have conducted extensive experiments using real system resource and workload traces collected on the production system. We learned the following lessons from our prototype implementation: 1) Real-world systems do exhibit significant patterns that can be captured by the system for more efficient load management; 2) Signature-driven load management can significantly improve resource utilization and QoS provisioning compared to traditional load management schemes. In our experiments, we observe that SigLM can improve system utilization by 30% to 80%; 3) Signature indexing can significantly speed up the signature pattern matching performance while maintaining the efficiency of the load management system; and 4) SigLM is feasible and efficient for large-scale distributed computing environments.

7 Acknowledgement

This work was sponsored in part by U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI), IBM Faculty Award, and NCSU startup fund.

References

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[2] Apache Hadoop System. http://hadoop.apache.org/core/.

[3] InfoScope Continuous Monitoring System. http://dance.csc.ncsu.edu/projects/infoscope.

[4] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. Patterson. Design and implementation tradeoffs for wide-area resource discovery. *ACM Transactions on Internet Technology*, 8(2):113–124, May 2008.

[5] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 15–15, Berkeley, CA, USA, 2004. USENIX Association.

[6] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. *OSDI*, 2004.

[7] M. Cardosa and A. Chandra. Resource bundles: Using aggregation for statistical wide-area resource discovery and allocation. In *Proc. of ICDCS*, 2008.

[8] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. *SOSP*, 2005.

[9] I. Foster. Grid: a new infrastructure for 21st century science. *Physics Today*, 2002.

[10] The STREAM Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin, 26(1):19-26*, March 2003.

[11] X. Gu, Z. Wen, and P. S. Yu. BridgeNet: An Adaptive Multi-Source Stream Dissemination Overlay Network. *Proc. of INFOCOM , 2586-2590*, 2007.

[12] James Patton Jones and Bill Nitzberg. Scheduling for parallel supercomputing: a historical perspective of achievable utilization. In *In Job Scheduling Strategies for Parallel Processing*, pages 1–16. Springer-Verlag, 1999.

[13] K.-L.Wu, P. S. Yu, B. Gedik, Ki. Hildrum, C. C. Aggarwal, E. Bouillet, W. Fan, D. George, X. Gu, G. Luo, and H. Wang. Challenges and Experience in Prototyping a Multi-Modal Stream Analytic and Monitoring Application on System S. *Proc. of VLDB, 1185-1196*, 2007.

[14] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Journal of Knowledge and Information Systems*, 2004.

[15] J. Kruskall and M. Liberman. The Symmetric Time Warping Problem: From Continuous to Discrete. *In Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison, pp. 125-161, Addison-Wesley Publishing Co.*, 1983.

[16] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. In *HotNets-I*, Princeton, New Jersey, October 2002.

[17] R. Raman, M. Livny, and M. Solomon. Policy driven heterogeneous resource co-allocation with gangmatching. In *Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 80, Washington, DC, USA, 2003. IEEE Computer Society.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *Proc. of the ACM SIGCOMM 2001, San Diego, CA*, 2001.

[19] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proc. of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[20] H. Shen and et al. Pird: P2p-based intelligent resource discovery in internet-based distributed systems. In *Proc. of ICDCS*, 2008.

[21] K. Shen, M. Zhong, and C. Li. I/o system performance debugging using model-driven anomaly characterization. *FAST*, 2005.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *Proc. of ACM SIGCOMM 2001, San Diego, California*, August 2001.

[23] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. *Proc. of SIGKDD*, 2003.