Achieving Scalable Automated Diagnosis of Distributed Systems Performance Problems

Chengdu Huang, Ira Cohen, Julie Symons, Tarek Abdelzaher
Enterprise Systems and Software Laboratory
HP Laboratories Palo Alto
HPL-2006-160(R.1)
June 1, 2007*

system performance diagnosis, machine learning, transfer learning, scalability

Distributed systems continue to grow in scale and complexity, resulting in increasingly more involved interactions among components and increasingly more intricate failure modes that are very hard to diagnose manually. This increased vulnerability of larger systems, together with the increased difficulty of failure diagnosis, has motivated machine learning approaches to automate the diagnosis task. While preliminary encouraging results are achieved, scaling up the existing approaches to large applications remains challenging. With increase in scale, current approaches suffer the curse of dimensionality exacerbated by the exploding set of system states and measured metrics. In this paper, we significantly improve scalability of performance diagnosis methods. Our contributions lie in the use of (i) an intelligent partitioning of the metric space, coupled with a cooperative *temporal segmentation* algorithm, dividing system observations in time and in space to remove the multiplicative explosion of system states, and (ii) *transfer learning* techniques that improve accuracy by leveraging dependencies among the partitions. We validate our approaches on several months of production traces from a customer-facing geographically distributed, 24 x 7, 3-tier internet service. Our results show a significant accuracy improvement (35% on average) over the naive partitioning of the state space (without the new temporal segmentation algorithm or transfer learning), and an order of magnitude reduction in computational cost over the "brute force" approach of learning with no partitioning, without loss of accuracy.

* Internal Accession Date Only
Approved for External Publication

Achieving Scalable Automated Diagnosis of Distributed Systems Performance Problems

Chengdu Huang†, Ira Cohen‡ , Julie Symons‡, Tarek Abdelzaher†
† {chuang30, zaher}@cs.uiuc.edu, University of Illinois at Urbana-Champaign
‡ {ira.cohen, julie.symons}@hp.com, Hewlett-Packard Laboratories

Abstract

Distributed systems continue to grow in scale and complexity, resulting in increasingly more involved interactions among components and increasingly more intricate failure modes that are very hard to diagnose manually. This increased vulnerability of larger systems, together with the increased difficulty of failure diagnosis, has motivated machine learning approaches to automate the diagnosis task. While preliminary encouraging results are achieved, scaling up the existing approaches to large applications remains challenging. With increase in scale, current approaches suffer the curse of dimensionality exacerbated by the exploding set of system states and measured metrics. In this paper, we significantly improve scalability of performance diagnosis methods. Our contributions lie in the use of (i) an intelligent partitioning of the metric space, coupled with a cooperative *temporal segmentation* algorithm, dividing system observations in time and in space to remove the multiplicative explosion of system states, and (ii) *transfer learning* techniques that improve accuracy by leveraging dependencies among the partitions. We validate our approaches on several months of production traces from a customer-facing geographically distributed, 24×7, 3-tier internet service. Our results show a significant accuracy improvement (35% on average) over the naive partitioning of the state space (without the new temporal segmentation algorithm or transfer learning), and an order of magnitude reduction in computational cost over the "brute force" approach of learning with no partitioning, without loss of accuracy.

1 Introduction

The complexity of current computing systems and applications is quickly outgrowing the human ability to manage it at an economic cost. It is common to find data centers with thousands of hosts serving hundreds to thousands of applications and components that provide web, computations and other services. In such environments, diagnosis of failures and performance problems is an extremely difficult task for human operators. To facilitate diagnosis, commercial and open source management tools measure and collect data from systems, networks and applications in the form of metrics and logs. However, with the large amounts of data collected, the operator is faced with the daunting task of manually going through the data, which is becoming unmanageable.

These challenges have led researchers to propose the use of machine learning and statistical learning theory methods to aid with the detection, diagnosis and repair efforts of distributed systems and applications[6, 7, 11, 12, 19]. In particular, our previous work [11, 12, 26] developed probabilistic models that associate low-level system metrics with application performance problems for a single instance of a 3-tier internet service.

In this work, we present scalable machine-learning-based techniques for diagnosis of performance problems in internet services that are composed of multiple instances (e.g., a distributed application replicated in multiple data centers). Each service instance may contain multiple servers and components such as 3-tier web services. Performance problems are defined through violations of service level objectives (SLOs). The SLOs define acceptable thresholds on performance attributes such as average transaction response times, the maximum number of allowable transaction failures in a given window of time, or combinations of such metrics. Following our prior work in [11, 26], the diagnosis task is to automatically point to the set of most indicative internal symptoms related to a given externally observed performance problem, detected as an SLO violation. These internal symptoms are a subset of continuously monitored data (comprising system metrics, application metrics and logs collected) that can point to the explanation of the performance problem. This automated diagnosis is achieved through learning probabilistic models capturing the correlation between the collected data and the SLO state.

Current automated diagnosis approaches work well on small applications but do not scale to very large distributed systems. The difficulties of scaling up the learning approaches stem from two reasons. First, with more components in a large distributed system, there is a large increase in possible causes of failure and hence an increase in the data measurements (i.e., metrics) that must be collected to pin-point them. These measurements together constitute a state space in which each individual metric is a dimension. Informally, regions in that state space must be identified that correlate with performance problems. Since the space grows exponentially in the number of metrics, more samples must be collected for the learning methods to populate the space sufficiently to identify "bad regions" with accuracy. Current methods suffer what is known as the "curse of dimensionality" [14]; a phenomenon in which they exhibit a reduction in accuracy for a fixed-size training set, as the number of metrics increases. With more data required and with the increase in the number of metrics, most learning methods can also become too computationally expensive. For example, based on our evaluation, using existing learning-based diagnosis techniques it takes over 30 hours of execution on a dedicated server to process a 30-day trace of an application with only 3 service instances (Section 4.3).

A second difficulty for learning algorithms lies in com-

bining different types of data, such as low level system metrics, application metrics, and semi-structured data (e.g., text based log files). The property that various types of data having different statistical characteristics (e.g., following different statistical distributions) making it challenging to combine them with existing learning methods.

In this work, we present automated diagnosis techniques that scale well to large-scale distributed systems containing many diagnostic information sources and replicated service instances. More specifically, we divide the wide range of metrics of different types (including both structured and semi-structured metrics) into bounded partitions to reduce learning overhead based on their semantics. A corresponding reduction ensues in the size of the training-set data needed to identify the problem regions. A key challenge in partitioning the metrics is to identify when a set of metrics from one source contain no diagnostic information for a given segment of the trace. We call this the *temporal segmentation* problem. For solving this challenge, we propose a method called *multi-source temporal segmentation* in which the inapplicability of metrics from a source on parts of the trace is detected by the existence of accurate models from other sources. This method significantly improves learning accuracy and has low-overhead. In addition, for service instances that belong to the same application, we take advantage of their similarities by applying transfer learning techniques, in which we allow models trained on one instance to be used on others. This enables further improvement in accuracy as well as leveraging existing diagnosis knowledge efficiently.

We validate our methods on several months of traces collected from a real geographically distributed and multi-instance 3-tier internet service. Our results show that our metric partitioning along with temporal segmentation and transfer learning approaches provide significant improvements in classification accuracy and retrieval of annotated performance problems over the naive method that ignores dependencies and similarities between the data sources or service instances. We also compare our methods to the brute force approach of no data partitioning, showing that our methods have much lower overhead but the same or higher accuracy.

The rest of paper is organized as follows. In Section 2 we present the problem statement and our approaches. In Section 3 we describe the traces we use to validate our methods. Section 4 provides use cases and empirical results demonstrating the validity of our approach. Section 5 describes related work. We discuss some open issues of our approaches in Section 6. The paper concludes with Section 7.

2 Problem Statement and Approach

This paper presents scalable learning-based problem diagnosis techniques for large-scale complex distributed systems. As systems grow in scale and complexity, we naturally need more diagnostic information sources to arrive at accurate diagnosis. In the view of learning algorithms, there will be more metrics. Learning from a large number of metrics simultaneously requires a significant amount of data and therefore a high computational cost. We call it the *brute force* approach. As scale increases, this approach may suffer a lower accuracy or longer "learning curve", in part, because it takes longer to observe all combinations of measured metrics (i.e., all system states) and correlate these combinations with good or bad behavior.

Dividing the metrics into smaller partitions and applying learning within each partition independently improves scalability. We call this the *naive partitioning* approach. This approach results in a different form of inaccuracy. Namely, it ignores dependencies between partitions. Additionally, it faces the issue of partitions that might not contain the right metrics to explain a given problem. Without the ability to detect this issue for a partition of metrics, the learning algorithm, which implicitly assumes that problems are correlated with (some) metrics, can produce inaccurate and unpredictable results.

The challenge addressed in this paper is to maintain the efficiency of the naive partitioning approach, while taking into account dependencies, as with the brute force approach. The resulting approach both achieves scalability and improves diagnosis accuracy. We begin by describing the brute force approach. It works well for smaller systems and is the starting point of our extensions. Subsequent sections describe our extensions then conclude with an architectural summary that presents the overall picture of our new scalable diagnosis method.

2.1 The Brute Force Approach

Our prior work on learning-based performance problem diagnosis [11, 26, 12] was shown to be successful for diagnosing problems on individuals instances of an internet service, using system metrics. The brute force approach is basically applying these approaches directly on all the metrics together. In this subsection, we briefly describe the prior learning methods.

In [11, 26], we automatically build probabilistic models that identify the set of metrics that correlate with each particular instance of the SLO state (compliance or violation). We use this information for constructing signatures that correctly characterize and distinguish different causes of SLO violations.

The methods work as follows. The input is a data log containing vectors $\vec{M}$ of measurements of system metrics and the state $S \in \{s^+, s^-\}$ (compliance or violation) of the system. For each regular epoch (e.g., 5-minute intervals) we have one such vector. Each element m_i of vector $\vec{M}$ for an epoch contains the value of the specific metric, and S contains a discrete value depending on whether the SLO was violated or not. Using pattern classification techniques, given a training window containing multiple epochs with both instances of violations and compliance, we learn probabilistic models [15] characterizing the behavior of a subset of the metrics that are most representative of the SLO state. A model is essentially a classifier function $\mathcal{F}$ mapping the universe of possible values for $\vec{M}$ to the range of system state: $\mathcal{F} : \vec{M} \rightarrow \{s^+, s^-\}$. Specifically, a model N represents the conditional distribution $P_N(S|\vec{M})$—the distribution of probabilities for the system state given the observed values of metrics. The classifier $\mathcal{F}$ uses this distribution to evaluate whether $P_N(s^+|\vec{M}) > P_N(s^-|\vec{M})$ to arrive at a prediction of the SLO state. The accuracy of a model at predicting the

SLO state is measured to establish the ability of the model to capture the service state.

With continuously collected traces, the algorithm in [26] produces an *ensemble* of probabilistic models. The ensemble is augmented dynamically with new models that are better at explaining the current problem. A model is good at explaining a problem if it can predict good and bad behavior with high accuracy given the measured metrics. Informally, a good model defines a cube with the state space of metrics that is highly correlated with bad behavior. The hope from constructing the ensemble is to arrive at a portfolio of models that can explain a large number of problems over time. Models that have not been good at explaining problems for a while are weeded out.

The current ensemble is used for describing the most related metrics to each SLO violation. Given a period of an SLO violation, s^-, the ensemble of models is used to identify which metrics (because of their values) are more likely to have been generated from their distribution during violation periods. This process is called *metric attribution*. Formally, for a given instance of SLO violation and each model in the ensemble, $P_{N_j}(\vec{M},S)$, a metric m_i is flagged as "attributable" if:

$$P_{N_j}(m_i|s^-) > P_{N_j}(m_i|s^+),$$

i.e., for model N_j, m_i's value is more likely to come from the "violation" distribution ($P_{N_j}(m_i|s^-)$) than from the "compliance" distribution ($P_{N_j}(m_i|s^+)$). This process hence identifies a subset of metrics that are the most relevant to the SLO violation according to the ensemble.

In [12], we have shown that we can use metric attribution for constructing signatures. These signatures describe the symptoms of SLO violations in terms of the metrics that are attributed (and those that are not); a signature is essentially a vector of attributions of the metrics. Since different instances of the same problem can generate models with slightly different parameters, the signatures are then subjected to automated clustering to group together those that likely describe the same problem. Different clusters can then be labeled by the problems they describe. When a new instance of a problem is observed and a new model is generated similarity-based retrieval can be performed to identify the nearest cluster and hence determine the (previously learned) problem. This allows operators to identify and quantify the frequency of recurrent problems and to leverage previous diagnostic efforts.

2.2 Scaling for Multiple Data Sources: Multi-source Temporal Segmentation

In our previous work, we have observed that there are periods of SLO violations for which no system metrics are attributed. This observation is not surprising—not every performance problem can be explained with system metrics. In general, the solution is to analyze different sources of data, such as application metrics, event logs (e.g., application errors, security, network), etc. Similarly, when more software components (such as additional database servers) are added, we also need to incorporate more data sources to maintain accurate diagnosis. As the number of sources of data increases, so does the number of metrics.

As the number of metrics increases, the brute force approach is to learn models with all metrics. Up to a certain number of metrics and system states, there are efficient algorithms that can produce results in reasonable time. To offset the "curse of dimensionality" and maintain accuracy, one would simply collect more training samples. However, we argue that such an approach, even if computationally feasible, will not produce accurate models. The reason is that traces collected over time are not stationary in various ways. For example, the traces contain different types of performance problems (with unknown number), and/or the underlying behavior of the application changes (e.g., due to configuration changes). Zhang *et al.* [26] demonstrated that as more types of performance problems are mixed in a training window, the accuracy of learning models decreases. The key challenge therefore is in segmenting the traces into the different regions representing either different types of performance problems or regions with no changes in the application behavior. However, as the number of metrics increases, this is a "chicken-and-egg" problem: with an increase in number of metrics, segments are required to contain more samples to avoid the curse of dimensionality. However, with an increase in number of samples in a segment, the chances of mixing different types of problems or different regions in application behavior in a segment also increases, leading to a loss of accuracy. Thus, to achieve scale while maintaining accuracy, the number of metrics considered in the learning over a training window needs to be bounded.

To enjoy the benefits of treating each source independently, while taking the best advantage of the different sources, we propose what we call *multi-source temporal segmentation*. The metrics are first partitioned based on topology and metric types, similar to the naive approach described above. We then automatically build an ensemble of models [26] on each partition as follows. Each ensemble maintains a training window which is a continuous sequence of training samples (measurements of the metrics). When a new training sample comes in, it is taken into the current training window. If the current training window contains enough samples of both flagged as SLO compliance and SLO violation, we perform a greedy selection feature selection [16] to pick the subset of metrics that is most relevant to modeling the SLO, and induce a probabilistic model [15] to capture the correlations between the subset of metrics selected and the SLO state.

The accuracy of models is then measured by balanced accuracy (BA) on the training window which is the average of the probability of correctly identifying compliance and probability of detecting a violation:

$$BA = \frac{1}{2} \times [P(s^- = \mathcal{F}(\vec{M})|s^-) + P(s^+ = \mathcal{F}(\vec{M})|s^+)] \quad (1)$$

Note that to achieve the maximal BA of 1.0, $\mathcal{F}$ must perfectly classify both SLO violation and compliance. If the new model has a high balanced accuracy in capturing the SLO state, and it is statistically significantly more accurate than the existing models in the ensemble, it will be added to the ensemble; otherwise it is discarded. Upon adding the new model, the ensemble will *reset* its training window. Moreover, the ensemble will instruct ensembles of other partitions (data sources) to reset their training win-

dow as well, if they have not been able to create an accurate enough model yet.

In this work, we use Naive Bayes models for the ensembles of each data source partition. Naive Bayes models are simple and efficient, and have sound semantics for producing metric attribution, a key feature required for explaining SLO violations.

The intuition behind this algorithm is to avoid model inaccuracies caused by training examples for which no metrics for a given data source are correlated with. In the context of server clusters, when training a model based on some data source, it is generally impossible to automatically discount SLO violation samples that do not affect this data source (i.e., for which no metrics of the data source are attributed). This is the segmentation problem mentioned earlier. The effect of including these samples in a training window is that they can skew the estimated statistics of metrics that otherwise would capture other periods of violation, leading to no models, or inaccurate ones. In our algorithm, the message from a learner of a data source indicating that it found attributable metrics for a given training window indicates to the learners of the other sources that the past epochs are of a problem that perhaps cannot be captured by that source, and therefore requires the collection of a new training window. One limitation of this approach is in cases when none of the collected data sources can produce an accurate model for some of the violations. However, as more data sources are analyzed, the odds of such occurrences is reduced.

It is worth noting that the method does not prohibit adding models from various data sources for roughly the same training window, as long as the models in these ensembles are accurate enough. In fact, the method can produce a combination of metrics from various sources that are attributed at the same time. In the production traces we collected, we did observe such cases.

2.3 Scaling for Multiple Instances: Transfer Learning

The second aspect of scaling the existing learning approaches is to accommodate *n*-replicated internet services, with each replicated instance containing all or some of the three tiers (web server, application server, database). These instances can be directly load balanced, or replicated across different data centers. Service instance replication is widely used in large scale distributed systems to improve throughput, reliability and dependability.

As the metrics from the different replicated instances are typically highly correlated, partitioning the metric space based on the topology of the service, i.e., to the different replicated instances, is an intuitively appealing heuristic, as it scales up with the number of instances (and metrics). Learning in this approach is performed with the metrics of each instance, independently of the other instances. However, this naive approach ignores the similarities between the instances.

Our approach is to leverage the similarities between different instances through the transfer of *models* between the different instances. The method works as follows: when the learner on one instance learns an accurate model on a training window for that instance, it transfers that model to all other instances of the internet service to be evaluated locally. A transferred model is used for attribution if it is deemed accurate. This transfer of models between different instances produces more accurate ensembles compared to the naive method of simply learning ensembles on each instance, ignoring the similarities and dependencies that exist between the instances.

Algorithm 1 Multi-source ensemble algorithm with temporal segmentation

```
Parameters: Minimum Number of Samples Per Class, Minimum Model Accuracy
Input: k data sources DataSource_{1,...k}
for each data source DataSource_i do
  initialize Ensemble_i to {φ} and TrainingWindow_i to {φ}
end for
for every new sample do
  for each data source DataSource_i do
    add sample to TrainingWindow_i
    if TrainingWindow_i has Minimum Number of Samples Per Class then
      train new Naive Bayes model M on TrainingWindow_i
      compute accuracy of M using cross validation
      if accuracy of M is higher than Minimum Model Accuracy and accuracy of M is significantly higher than the accuracy of all models in the Ensemble_i then
        add M to Ensemble_i
        reset TrainingWindow_i to {φ}
        notify other data sources
      end if
    end if
    if receive notification from any other data source then
      reset TrainingWindow_i to {φ}
    end if
  end for
end for
```

Transferring models is a form of transfer learning [24]. The intuition behind transfer learning theory is that it is possible to transfer what is learned about one classification problem to related ones. When transfer learning is possible, it reduces the amount of training examples the learner needs to observe to obtain accurate classification models. In the case of multiple instances of an internet service, the similarities are in the fact that the instances display similar behavior. Transferring models between the instances reduces the amount of samples required to be seen on each instance of a particular problem, since if that problem was already previously observed on another instance and produced a model, it is directly and immediately applied. Our results (in Section 4) show significant improvements in the ensemble accuracy using this transfer learning approach compared to the naive method.

In addition, we hypothesize that transferring models also produces more consistent signatures (Section 2.1) of similar performance problems between the different instances, leading to improved retrieval of signatures across different instances. Our empirical analysis supports this hypothesis (Section 4).

The added complexity in transferring models in minimal: models have very small footprint ($<$ 1KB) and evaluation of a model on an epoch takes few milliseconds. One re-

quirement for transferring models is that the mapping between the collected metrics between the different instances be known (e.g., application server CPU utilization metrics are mapped to each other, even if they are named differently on each system). This requirement is easily met when the same data collection tools are used on all instances (such as HP OpenView).

2.4 Information Extraction From Event Logs

Unlike the system metrics and application metrics, which consist of structured numeric data, the application event logs are semi-structured and contain free text information. The event logs are essentially messages written by the developers of the application. There are potentially many different messages. For example, in the logs collected on one instance of FT system in a 9-month period, there are more than 280,000 distinct messages (after removing timestamps and fields containing numerical symbols only). Hence, we need to distill the smaller set of "prototypical" feature messages from the event logs. With this set of feature messages defined, we count the number of times each feature message appeared in a given time interval (set to match the interval of the SLO metric), and use these counts as the input metrics for the learning algorithms. In this section, we present a novel algorithm for sequentially and efficiently distilling the prototypical feature messages from text logs.

A natural approach to distill prototypical feature messages is to perform text clustering [25]. Messages that are similar enough will be combined to form a cluster. For example, messages generated by the same `fprintf` statements with slightly different parameters could probably be clustered. Basically, message clustering reverse engineers the "templates" generating these messages and ignore the minor differences. While text clustering has been extensively studied in literature, a unique challenge in our scenario is that the clustering must be performed in an incremental fashion because over the lifetime of the system, several code changes are pushed into production and new messages appear. It is infeasible to wait until all possible messages are seen in collected logs before they are clustered.

We developed a similarity based sequential clustering algorithm. We measure the similarity between two text messages with the cosine distance:

$$D_{cos}(A,B) = \frac{\sum_i match(a_i,b_i)}{\sqrt{|A| \cdot |B|}}, match(a_i,b_i) = \begin{cases} 1 & \text{if } a_i = b_i \\ 0 & \text{otherwise} \end{cases}$$

where A and B are the messages, $|\cdot|$ represents the number of words in a message, and a_i is the i'th word in message A. The cosine distance is a number between 0 and 1. When $D_{cos} = 1$, the two messages, A and B, are identical, and when $D_{cos} = 0$, the two messages are completely different. Upon seeing a new message, the clustering algorithm compares the message with the existing clusters. If there exists a cluster to which the cosine distance is larger than a predefined threshold (we used 0.85 in our experiments), then the message will simply be merged to the existing cluster. Otherwise, a new cluster will be created using this new message. For example, messages

- `java.net.connectexception: db server connection refused; error host001`
- `java.net.connectexception: db server connection refused; error code`

are clustered together because their cosine distance is 0.857 (> 0.85).

This method is simple, efficient, and does clustering incrementally. Empirically, we found that it significantly reduces the number of distinct messages (to clustered prototypical messages), and yields clusters with good quality. There are, however, limitations and issues with this simple approach such as information loss caused by clustering and too many distinct messages for large systems. We shall discuss these issues in Section 6.

As stated earlier, with a small number of feature messages extracted from the raw logs, we then count the appearances of the feature messages during 5-minute intervals, and use the counts as the metrics to learn the ensemble of models. It is worth noting that the statistical properties of these feature message based metrics is different compared to system utilization based metrics or application metrics. Indeed, in applying our methods on these metrics we use a different distribution in the probabilistic models. For system metrics we use the normal distribution, while for message based metrics we use a modified Gamma distribution, which we observed to fit better than the normal and other distributions. Formally, the modified Gamma distribution follows

$$P(x = X) = \begin{cases} p_z & \text{if } X = 0 \\ (1-p_z)X^{k-1}\frac{e^{-X/\theta}}{\Gamma(k)\theta^k} & \text{otherwise} \end{cases}$$

Note that x is always a non-negative integer. The modified Gamma distribution fits the feature message counts better because these counts exhibit a heavy tail with an additional large concentration of 0 counts.

2.5 Summary of System Architecture

Figure 1 depicts the software architecture of our automated diagnosis solution. Each service instance may contain multiple servers and software components (e.g., a 3-tier web service). These servers and components are instrumented to measure a wide range of metrics, collect various kinds of event logs, as well as monitor the service SLO state. Semi-structured data such as text-based logs are processed using the algorithm describe in Section 2.4 to distill diagnostic information. These metrics coming in periodically are first partitioned, as mentioned earlier, based on knowledge of topology of the components and type information of the metrics. The partitioning yields multiple metric partitions (e.g., system metrics, application metrics, and event logs), each having a bounded number of metrics.

With the metric partitions and the monitored SLO state, we automatically learn an ensemble of models for each partition using the multi-source temporal segmentation technique (Algorithm 1). Metric partitions that successfully generate accurate enough new models instruct other partitions to reset their training windows. This mechanism helps the ensembles to better divide the continuous measurements into training windows to facilitate generating accurate models. Furthermore, transfer learning techniques are applied by exporting the to the ensembles using the same metric partition on other service instances (e.g., replicated service in other data centers).

Using the ensembles of models, metric attribution is per-

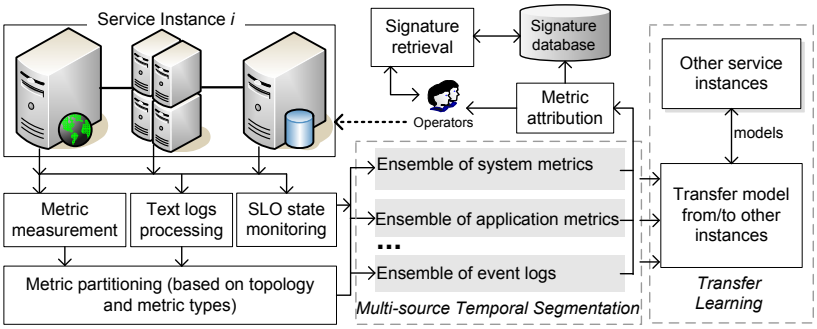


Figure 1: System architecture of our scalable diagnosis solution. Various types of metrics from different components are divided into partitions. Ensembles are built on each partition and influence each other using the multi-source temporal segmentation algorithm. Models induced on one service instance can be transferred to other instances of the application. All models in the ensembles contribute to the metric attribution which generates signatures for the SLO violations. The signatures help operators with troubleshooting.

formed for each SLO violation epoch. As there are multiple models in an ensemble, the models are fused for predicting the SLO state using the Brier score [26, 8, 10]. The Brier score is the mean squared error between a model's probability of the SLO state given the current metrics and the actual value of the SLO state. Formally, for every model N_j in the ensemble, on a short window of recent data $D = \{d_{t-w}, ..., d_{t-1}\}$,

$$BS_{N_j}(D) = \sum_{k=t-1}^{t-w} [P_{N_j}(s^-|\vec{M} = \vec{d_k}) - I(s_k = s^-)]^2 \qquad (2)$$

where $P_{N_j}(s^-|\vec{M} = \vec{d_k})$ is the probability of the SLO state being in violation of model N_j given the vector of metric measurements d_k and $I(s_k = s^-)$ is an indicator function, equal to 1 if the SLO state at time k is s^- and 0 otherwise. The attributions from all models in the ensembles are filtered by using only the most accurate models in terms of their Brier score on the most recent samples prior to the violation. Similarly, for models transferred from other instances, they can be used for metric attribution when their Brier scores are good.

This process generates signatures (represented as vectors of metric attributions) for each violation epoch. Signatures are stored in a database and can subject to clustering and similarity based retrieval [12]. The subset of metrics attributed by the ensemble allow the operators to focus on a small number of possible troubleshooting options. The signature retrieval further enables leveraging previous diagnosis efforts [12].

Note that our solution does not limit to the three metric partitions shown in Figure 1. Our data partitioning and transfer learning techniques are highly scalable and can support the system to learn with a large number of data sources without loss of learning accuracy.

3 Trace Collection and Processing

Our empirical results are based on detailed traces collected from a distributed application in a globally-distributed production environment. The application, called "FT" for confidentiality reasons, serves business-critical customers across the globe 24 hours per day, 365 days per year. Its system architecture therefore incorporates redundancy and failover features both locally and globally. FT is distributed across three regions: Americas, Asia, and Europe. Each region consists of a 3-tiered architecture that includes multiple client web front ends, 3 application server instances, and 2 backend database servers.

Region	Average Transactions per min	Average Response Time	SLO Violation Percentage
Americas	57.1	2.35s	14.6%
Asia	27.7	3.84s	16.9%

Table 1: Summary of FT application traces. Trace collections cover 3 months for the 3 Americas and 2 Asia application instances. This data represents averages over each instance in a region. The final column is the percentage of 5-minute windows with SLO violations.

3.1 Trace Characteristics

The traces contain various types of data about the application. These data types comprise the different data sources. There are system-level metrics, including resource utilization measures (e.g., CPU utilization) from both the application and database tiers. There are application event and error logs. Also, there are application-level performance data, including volume, response time, and failure counts for each transaction and sub-transaction, aggregated over 5 minute windows. From key metrics in the application-level data, SLOs are defined. The criterion for SLO violation is whether the average response time over all transactions in a 5-minute period exceeds 3 seconds or the transaction failure count is greater than 20.

HP OpenView Performance Agent (OVPA) provides system-level metrics for application server and database hosts. The FT application is instrumented using ARM [23] to provide the application-level metrics. OVPA and ARM data are aggregated into 5-minute windows. The application event logs come from each of the application server instances and are in text form.

The FT traces are summarized in Table 1. The transaction volumes seen in the traces demonstrate the non-trivial workloads of the FT installations. We have system and application measurement data for all three Americas instances and two Asia instances, and event logs for the America instances.

Some of our 5-minute samples exhibiting performance problems are "annotated". They correspond to times when we know, based on operators' troubleshooting documentation, that a specific performance problem occurred whose root cause was subsequently diagnosed. In the traces used in this paper, annotations are provided to seven periods, ranging from 2 to 15 hours of SLO violations. The causes of these problems included both local (file system full in one case), and external (availability problems of another service on which the FT service relies for some transactions) components.

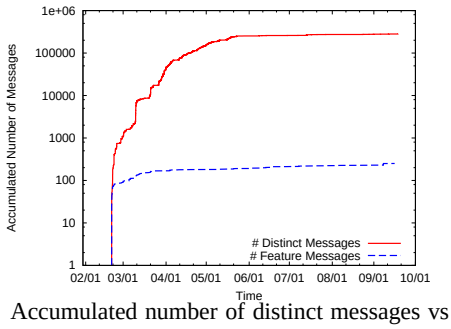


Figure 2: Accumulated number of distinct messages vs number of feature messages

3.2 Event Logs Processing

We use the clustering algorithm for information extraction from text-based logs described in Section 2.4 to process the event logs of the FT service instances. Figure 2 and Table 2 summarize the performance evaluation of our clustering algorithm. Figure 2 plots the accumulated number of distinct messages and number of feature messages (clusters) over time for one of the instances of FT. As can be seen in the figure, while there are a large number of distinct messages in the event logs, our clustering algorithm successfully clusters them to a small number of feature messages: for this data set, there are 212 feature messages, compared to 281,405 distinct messages.

To understand how good the clustering results are, we validated our sequential algorithm against a widely used clustering algorithm called Hierarchical Tree Clustering, which operates on the batch of data, rather than sequentially. We compare the clustering results of our sequential clustering algorithm and that of Hierarchical Tree, and measure the difference using a metric in literature [17]. This metric is a real number in the range of $[0,1]$; 0 indicates an exact match of the two and 1 indicates they are complete different. From our experiments, the difference for the logs on the three instances are all very small, and the average is 0.067.

	Instance1	Instance2	Instance3
Total Days of Logs	125	204	121
Total Size of Logs	2.5GB	3.5GB	2.1GB
# of Log Entries	4,989K	7,262K	3,887K
# of Distinct Msgs	43,321	281,405	46,203
# of Feature Msgs	197	212	177
Processing Time	281 sec	540 sec	287 sec

Table 2: Event logs of three FT instances

4 Results

We present a performance evaluation of the techniques we propose, namely multi-source temporal segmentation for learning using multiple data sources, and transfer learning for scalable diagnosis with multiple service instances. We use the traces collected from the FT service, described in the previous section. We validate that our techniques achieve scalability and improve accuracy of the diagnosis.

We demonstrate the success of our methods with classification accuracy, retrieval accuracy, efficiency analysis and through detailed case studies.

The classification accuracy of ensembles in predicting the SLO state is our first measure of success. Classification accuracy is a proxy for the usefulness of our diagnosis system since it captures how well the models describe the SLO state on all the data. Although a high classification accuracy does not immediately translate to high diagnosis accuracy, it serves as strong evidence that the ensemble can have better diagnosis performance. Plus, it has the advantage that it can be applied to all of the data. In our problem, accuracy comprises of (1) *SLO violation detection rate*, defined as the percentage of SLO violations that are correctly identified by the ensemble; and (2) *False alarm rate*, defined as percentage of SLO compliance samples that are falsely identified as violations by the ensemble. In all of our experiments, we compute the classification accuracy *sequentially*: given a new sample at time t, the existing ensemble before t is used to predict the SLO state, after which the sample is potentially added to a training window. This ensures that there is no overlap between the training set and test set used to compute the classification accuracy of an ensemble. It also produces a time series of accuracy, which provides additional insights into changes in behavior of the service, as we demonstrate in the case studies described in the following sections.

A second measure of success in comparing the methods is the retrieval accuracy of annotated problems using the signature. Annotated problems are those that have known problem types labeled by the operators. Retrieval accuracy measures the capability of identifying similar performance problems using the signatures generated by metric attribution with the ensemble we build. While this measure is the most accurate measure of success for diagnosis, it hinges on the availability of accurately annotated problems, which are difficult to obtain in most traces, as operators do not diagnose every problem, and do not always diligently record their actions. We report retrieval accuracy for a subset of the performance problems in our traces for which annotations were provided by the operators of the service.

We also provide detailed case studies, analyzing the usefulness of our methods for diagnosing specific incidents that appeared in our traces.

4.1 Multi-source Temporal Segmentation

We begin by evaluating our learning method for combining multiple data sources with temporal segmentation. We provide separate results for each of the three Americas instances of our application and three data sources collected on each.

4.1.1 Ensemble Accuracy

We start with evaluating the accuracy of ensembles obtained from individual data sources. Since an ensemble contains multiple models, we use the weighted vote (using Brier score as the weight) of the three best scoring models (using the Brier score) to arrive at a single prediction of the SLO state (compliance or violation). Figure 3 shows the *online accuracy* (broken into violation detection rate and false alarm rate) of the ensembles for one instance of the FT service. A separate curve is shown for each of three data sources; namely, system metrics, application metrics, and event logs. These sources are used in isolation. We make three observations here. First, we see how the violation detection rates increase and decrease at certain periods. An increase follows the addition of each new model to

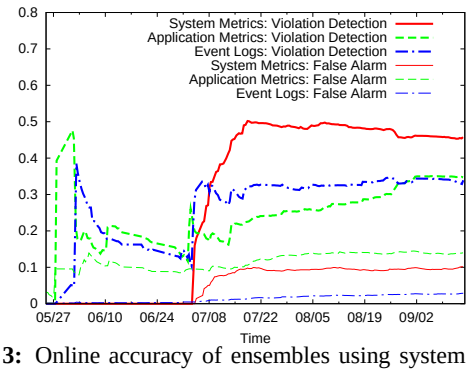
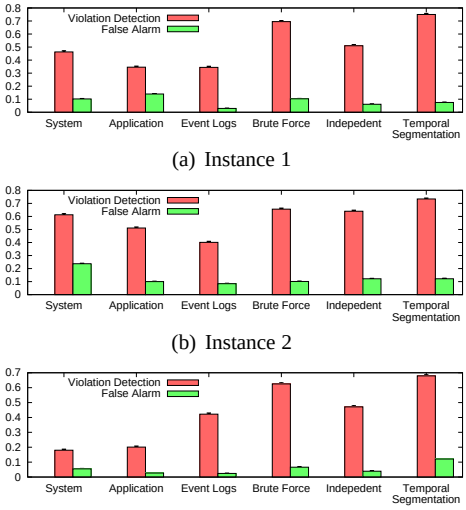


Figure 3: Online accuracy of ensembles using system metrics, application metrics, and event logs.

the ensemble. A decrease occurs when no accurate model exists or can be trained in the ensemble. The curves gradually stabilize after a sufficient amount of time. This is partly because the online accuracy is the aggregate of all the previous samples and hence impact of erroneous predictions diminishes as number of samples increases. In addition, as more models are added to the ensembles, they accommodate more types of problems, and therefore avoid fluctuations in the detection rate. Second, while overall the ensemble using just system metrics has a higher detection rate than the other two, over the entire period, we see periods where one data source outperforms the others. The implication of this observation is that some combination of the multiple data sources should improve overall accuracy. Finally, we see that the detection accuracy of individual sources is not very high. This observation further motivates combining multiple sources.

Next we compare our multi-source ensemble learning algorithm (Algorithm 1) with the two baseline algorithms discussed in Section 2: the unscalable brute force method that combines all metrics from all sources together to form one single ensemble, and the efficient method which generates an ensemble independently for each source. We hereafter call them "brute force" and "independent", respectively. Note that the brute force method generates one ensemble, while independent and temporal segmentation generate one ensemble for each data source. Similar to the case of a single ensemble, to obtain a single prediction of the SLO state with multiple ensembles, we use the weighted vote (using Brier score as the weight) of the three best scoring models (using the Brier score) from all ensembles.

Figure 5 shows the online accuracy for the three different multi-source methods on a single FT instance. First, compared to the online accuracy curves in Figure 3, we see that all three methods produce more accurate predictions compared to the accuracy of any single source individually. Among the three multi-source methods, our temporal segmentation algorithm clearly reaches a much higher violation detection rate than the others, while the false alarm rates of the three are roughly the same. On average, temporal segmentation achieves a 35% improvement over the independent method in SLO violation detection rate. Interestingly, our algorithm outperforms even the brute-force (much more computationally intensive) approach of not partitioning the metric space. This is probably due to the curse of dimensionality: there is simply not enough data to obtain good models in such a high-dimensional space.



(a) Instance 1

(b) Instance 2

(c) Instance 3

Figure 4: Overall ensemble online accuracy of single-source ensembles and multi-source ensembles using different methods, for two FT service instances. Accuracies are provided with their 95th percentile confidence intervals.

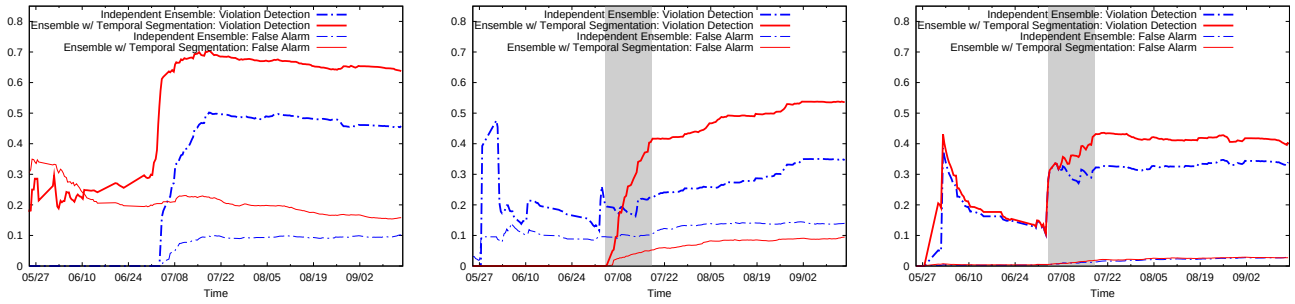
Additional results for the other two FT instances are summarized in Figure 4. We again see that accuracy using all multi-source methods is significantly better than any single individual source. In addition, we observe significant violation detection improvements with our temporal segmentation method, although the method exhibits slightly higher false alarm rates in some instances. Note that even with these data sources, some SLO violations are not captured (the best violation detection is 75% in Figure 4). Other sources of data would be required to capture those SLO violations, which in turn would aggravate the scalability problem of the brute force method.

Further demonstrating the advantage of our temporal segmentation method, Figure 6(a-c) shows the online accuracy curves for each source, comparing the ensemble for that source generated with our method (passing messages between the different sources), and the independent method of generating the ensemble for that source. The curves for the ensembles of the independent method are basically the curves shown in Figure 3. Note that in this experiment we do online testing on the ensembles for each data source generated by the temporal segmentation method as if they are separate ensembles. The purpose of this experiment is to demonstrate that temporal segmentation not only improves the accuracy of the combined ensemble (as shown in Figure 5), but also improves accuracy of individual ensembles.

As can be observed in the figures, all ensembles of our temporal segmentation method achieve better violation detection than the purely independent method, with the exception that the ensemble of system metrics suffers a slightly higher false alarm rate.

4.1.2 Metrics Attributed

We present statistics of metrics attributed by the ensembles built with multi-source temporal segmentation in Table 3. In the table, we give the total number of metrics and metrics that are attributed by the ensembles for each data source and service instance. Note that the numbers of met-



(a) System Metrics (b) Application Metrics (c) Event Logs

Figure 6: Accuracies of ensembles created for each data source using independent learning approach and temporal segmentation approach.

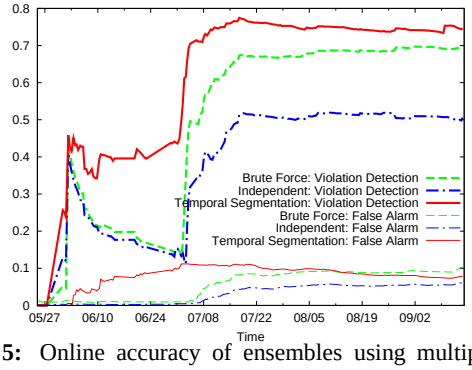


Figure 5: Online accuracy of ensembles using multiple data sources. Ensemble with temporal segmentation achieves better accuracy than brute force and independent.

		System	Application	Event Logs
Inst1	Total	55	25	197
	Attributed	22	10	23
Inst2	Total	55	25	212
	Attributed	16	9	24
Inst3	Total	55	25	177
	Attributed	15	9	21
Shared	Attributed	8	7	19

Table 3: Metrics attributed for different data sources and service instances by ensembles built with multi-source temporal segmentation for the 3-month trace.

rics attributed shown in the table are *not* for a single SLO violation. Instead, they are the union of metrics attributed for all the SLO violation epochs during the 3-month period. In fact, on average 4 metrics are attributed for each SLO violation.

As we can see, overall a significant portion of the metrics (17-20%) are attributed for some of the SLO violations, signifying that the system experienced many different types of performance problems. Another observation is that the set of metrics attributed by different service instances are not exactly the same (number of shared metrics are shown in the last row of the table). This suggests that during the period of the traces some problems occurred in some instances but not the others. Therefore, transferring the models for those problems (Section 2.3, 4.2) will be useful for the other instances in the future when they encounter those problems.

4.1.3 Case Study 1

We present a case study observed in our traces that further illustrates the benefits of both leveraging multiple data sources and the efficacy of our temporal segmentation technique. Specifically, we describe a performance problem appeared in the traces that using multiple data sources provides more meaningful diagnostic results than single source, and the temporal segmentation technique enables the generation of a very useful model.

Our traces contain a period in which a disk on one of the application servers for one of the FT instances became full. The full disk caused the failure of a component to launch (CORBA wrapper) because it could not write some necessary files to disk. This component failure caused high response times and high transaction failure counts because some transactions were hung, eventually timing out, waiting on responses from the failing components.

The ensemble trained using event messages quickly created a model that attributed the following message: `CORBA access failure: IDL HPSE wrapper can not start`. The ensemble using system metrics generated a model using the file space utilization metric (which, in this case, indicated that the file system was full). Obviously, the combination of these sources provides a much clearer diagnostic picture to the operator. This example shows the need to use multiple data sources for accurate diagnosis, as it reveals to the operator not only that the file space utilization is attributed (and full), but also the effect of that system condition on the application (failure of a component to launch). However, capturing these relevant metrics did not require the pooling of all metrics to create a single ensemble.

This problem was in fact a recurring problem. It happened once before for a short duration (1.5 hours), which was not sufficient for creating a model. The problem occurred again for a duration of over 10 hours about 10 days after the models were created. This time, the ensemble captured the problem and attributed related metrics correctly.

Additionally, certain application metrics were also related to the problem (namely, the value of some of the transactions counts). With our temporal segmentation algorithm, as the ensembles using system metrics and event logs generated new models during this time period, the ensemble using application metrics reset its training window to start collecting samples from this period. It then created models with the related application metrics. The models generated turned out to be very helpful in predicting SLO state, as can be seen in the large jump in detection accuracy for the application ensemble marked in Figure 6(b). In contrast, the independently trained ensemble with application metrics did not produce any model for this period of violation, because it failed to reset its previous training window

(which contained periods of violations that were not related to the problem). As can be seen in Figure 6(b), without any model that captures this problem, the detection accuracy of the ensemble declines and remains much lower compared to the ensemble trained with temporal segmentation.

4.2 Evaluation of Transfer Learning

We now evaluate applying our transfer learning technique (Section 2.3) on multiple replicated FT service instances. We study the improvement in terms of ensemble accuracy and signature retrieval performance.

4.2.1 Ensemble Accuracy for Load-Balanced Instances

We start with studying the efficacy of model transfer between geographically co-located load-balanced instances. These instances are most similar in the sense that they have similar workloads, run on similar hardware and exhibit similar problems when the source of the problem is non-instance specific. In this experiment, we use the three FT instances in America. These instances have similar hardware configurations, and are load-balanced. We apply model transfer to learning with a single data source for the sake of separating concerns. We then apply it to learning with multiple data sources to understand the composability of our techniques for multi-source and multi-instance learning.

Figure 7 shows the online accuracy of one instance when applying model transfer with an ensemble based on application metrics. We marked some of the periods when models were transferred from other instances. As we can observe from the figure, initially, the learning algorithm was not able to train good models, and hence had a nearly zero violation detection rate. However, eventually, models transferred from other instances successfully identified a significant portion of the SLO violations. Overall, the comparison shows that transfer learning significantly improves the accuracy in this case.

Figure 8 plots the online accuracy of one instance when model transfer is applied together with multi-source temporal segmentation, compared to that in the absence of transfer learning. Recall that, in Section 4.1, we demonstrated that our multi-source temporal segmentation technique greatly improves accuracy over single data source and other multi-source approaches. Here, as can be observed from the figure, transfer learning further achieves appreciable improvement in terms of accuracy.

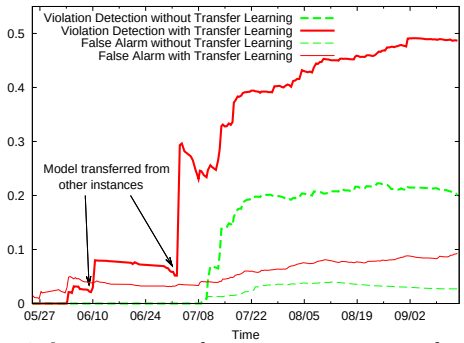


Figure 7: Online accuracy of an instance using transfer learning on application metrics.

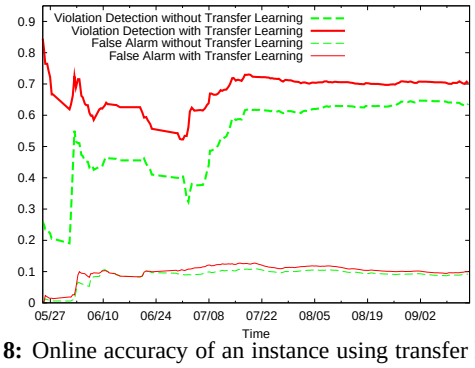


Figure 8: Online accuracy of an instance using transfer learning on multiple data sources with temporal segmentation.

Evaluation results for applying model transfer technique on other data source and instances are summarized in Figure 9. Compared to the data reported in Figure 4, we observe that for single-source ensembles, transfer learning consistently improves ensemble accuracy. The improvement is substantial in some cases (e.g., on Instance 3). In terms of multi-source ensembles, the improvement of using transfer learning is not as prominent, but never has significant negative impact either. We conjecture that for load balanced instances, the overall gain in accuracy is not always significant because it is likely that all instances suffer from similar problems at the same time, if those are related to workload or availability problems of shared resources (e.g., network, auxiliary databases). However, the overhead of transferring models and adding models to an ensemble is small, making it always worth it to perform transfer of models.

4.2.2 Ensemble Accuracy for Instances Across Data Centers

Next, we apply our transfer learning technique to service instances across geographically distributed data centers. In the FT service, we have system metrics and application metrics measurement data on two of the instances in Asia[1]. We report our experience of transferring models trained on the instances in America to instances in Asia in Table 4. As the tables show, transfer learning improves violation detection across all of the cases. For the ensemble of system metrics on Inst2, we see a substantial improvement. In terms of false alarms, transfer learning slightly reduces false alarms in most cases.

	Data Source	Vio Detection	False Alarm
Inst1	System	0.744 ± 0.009	0.154 ± 0.004
	Application	0.625 ± 0.008	0.078 ± 0.004
Inst2	System	0.629 ± 0.008	0.101 ± 0.004
	Application	0.701 ± 0.008	0.131 ± 0.004

(a) Without Transfer Learning

	Data Source	Vio Detection	False Alarm
Inst1	System	0.779 ± 0.009	0.087 ± 0.003
	Application	0.655 ± 0.008	0.056 ± 0.003
Inst2	System	0.786 ± 0.008	0.164 ± 0.004
	Application	0.719 ± 0.008	0.101 ± 0.004

(b) With Transfer Learning

Table 4: Ensemble accuracy of FT instances in Asia with/without models transferred from instances in America.

Note that although the instances in Asia and America are

[1]Unfortunately, we do not have event logs of these instances.

(a) Instance1 (b) Instance2 (c) Instance3

Figure 9: Overall ensemble online accuracy with transfer learning applied on load-balanced instances.

replicas of the same service, they run on different hardware configurations and are subjected to different workloads (as users of the service in Asia have a different usage profile of the service). However, despite these differences, our transfer learning method exhibits the ability to improve ensemble accuracy. In fact, we observe higher accuracy improvements, when transferring models across data centers, compared to load-balanced instances in the same data center. We hypothesize that because the former are operationally more independent (but exhibit similar problems), they train models for those problems at different periods, hence leveraging each other's models more often.

4.2.3 Retrieval Accuracy

Another important benefit of transfer learning is that through transferring models, instances end up with a more homogeneous set of models. This is important for diagnosing problems that are shared by multiple instances. Obviously, if a common problem occurred on multiple instances, we would like a similar set of metrics to be attributed on all these instances. Hence, when it came to diagnosis, one would easily identify that multiple instances had the same problem. In practice, however, instances train models independently. This can lead to different instances attributing a slightly different set of metrics because there could exist multiple metrics that capture the problem. Using model transfer, however, can alleviate this phenomenon. When a model trained on one instance is transferred to other instances, it will be used to attribute the problem it was trained for on other instances, given it is accurate enough (with high enough Brier score). Therefore, the probability that the same problem will be attributed with the same set of metrics is enhanced, which is very important for diagnosing large scale systems.

We evaluate this advantage of our transfer learning technique through signature retrieval [12]. Using metric attribution, we generate signatures [12] SLO violation epochs, and store them in a database for retrieval. The process of retrieval proceeds as follows: given a signature, return the N closest signatures to it from the existing signature database. Retrieval accuracy measures the ability of using the signatures to accurately identify problems of the same type. Formally, given known annotations both to the query signature and the signatures in the database, we compute the two standard measures of retrieval quality: Precision and Recall [25]. Precision measures what fraction of the N returned items have the matching annotation (1.0 is perfect); recall measures the percentage of signatures in the database with the same annotation as the query that are actually retrieved. As N increases recall goes up but precision typically goes down, as it becomes harder to retrieve only signatures that have a matching annotation. Following the

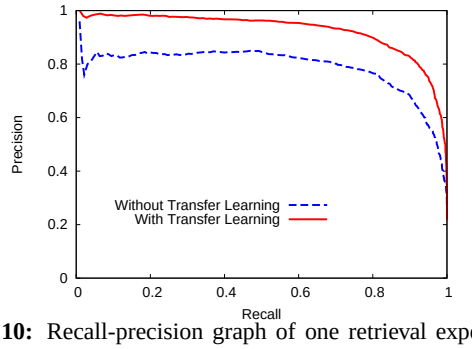


Figure 10: Recall-precision graph of one retrieval experiment. Model transferring improves retrieval precision.

common practice in the information retrieval community, we increase N and measure the precision/recall pair, until we achieve a recall of 1.0. We then plot precision as a function of recall, to produce the Precision-Recall curve. A perfect precision/recall curve has precision of 1.0 for all values of recall.

During the period that we have measurement data available, there are over 700 annotated SLO violation epochs on the three America instances for 3 different types of recurring performance problems. We use signatures generated on one instance for these problems to retrieve annotated signatures in other instances, and repeat this for every instance. It is worth noting that the 3 types of problems are just a small part of the all the problems experienced by the instances: only 700 epochs out of 11,000 epochs are annotated by the operators. Figure 10 plots the Recall-Precision curves of one of our experiments. We use multi-source ensembles, and compare the retrieval precision with and without transfer learning applied. From the figure, we can see that both methods achieve high precision, which is related to the fact that our multi-source ensembles have very good accuracy. Comparing the two, model transfer has clear advantage compared to when transfer learning is absent.

To measure the overall retrieval performance, we use an aggregate metric called AUC [21], which is the area under the recall-precision curve. The value of AUC is $[0,1]$, with 1 being the best possible retrieval. We further aggregate these AUC over all our retrieval experiments and use it as the indicator for overall retrieval performance. The aggregated AUC of using multi-source ensemble without transfer learning is 0.7656. With transfer learning the AUC is 0.8587, a significant improvement.

4.2.4 Case Study 2

The intuition behind our transfer learning technique is that since replicated service instances are similar in functionality (and architecture), it is likely that the instances will experience roughly same set of problems. However, they may not always experience the same problem at the same

time. Furthermore, even when a new problem emerges on multiple instances at the same time, it is not necessarily the case that all the instances would be able to train good models for the problem. Transferring models can help instances to quickly identify performance problems and arrive at correct metric attributions.

During the period reflected in our traces, there was a recurring problem with timeout on accessing an auxiliary database. This database is shared by the instances in America. In early March, Instance2 experienced this problem for a period of several contiguous hours. Instance2 hence trained a good model of this problem. At roughly the same time, Instance1 also experienced the same problem, but with a much smaller scale—only 10 sporadic epochs. This was related to the type of transactions the instance processed during that period.

When training each instance independently, since there were not enough SLO violations on Instance1, it did not generate a model for this problem. It was not until late August that Instance1 successfully generated the model when the auxiliary database had a similar problem for a sufficiently long time. However, for 90 epochs (a total of 7.5 hours) in which the problem occurred prior to learning that model, Instance1 had no accurate model capturing them.

In contrast, when the transfer learning was used, Instance2 transferred its model for the database timeout problem to Instance1 immediately after the model was created. With this model, Instance1 was able to identify the problem, and correctly attribute related metrics. In fact, with the model transferred from Instance2, Instance1 successfully identifies and attributed 91% of the 90 epochs which were missed with independent learning.

4.3 Efficiency and Scalability Evaluation

The previous results establish the ability of our methods to produce accurate models and diagnosis. Next, we present computational cost for different learning methods in our diagnosis framework. The computational complexity of most learning algorithms does not grow linearly with the number of metrics. For example, a simple greedy feature selection algorithm [16] would be $O(n^2)$. Other methods can be more expensive. As the number of metrics increases, our method of partitioning the metrics into small and bounded subsets is expected to be significantly more efficient than the method using all of the metrics.

In Table 5 we give the ensemble training time for a 30-day trace, with samples every 5 minutes. The experiments were run on a Pentium4 3.5GHz PC with 1GB memory.

From the table, it is clear that using all metrics together for learning incurs a higher computational cost than partitioning the metrics. While the absolute run time for the brute force method on a single instance is not prohibitive, it has very poor scalability. When the brute force method was run on combined metrics from the three instances, it took about 1.5 days to train models for the 30 day trace. In contrast, our partitioning techniques manages to keep the run time within a few hours.

Note that our trace is from a relatively small scale application. Based on our personal communication with a large scale internet service, large scale internet services can have over a million measurement metrics. For systems of that scale, the brute force approach would simply break down.

		# metrics	run time
single-source	System	55	6min
	Application	25	4min
	Event Logs	212	73min
multi-source	Brute Force	292	145min
	Independent	292	83min
	Temporal Segmentation	292	86min
multi-instance	Brute Force	826	2018min
	Independent	826	252min
	Temporal Segmentation w/ Transfer Learning	826	270min

Table 5: Ensemble training time of different methods. Partitioning data (independent or temporal segmentation) significantly reduces run time.

Besides the computational advantages of partitioning the metric space, the actual learning can be distributed to several machines, or done locally on servers of each instance. The communication overhead for distributing the learning is very low. The messages passed with our multi-source temporal segmentation algorithm are just a few bytes (id of source, time stamp, model flag), and the typical model transferred in our transfer learning algorithm is no more than 1KB in size. Similarly, composition of signatures and retrieval can be done in a distributed fashion, achieving much higher efficiency over central analysis.

5 Related Work

Because our research is multi-disciplinary, this section surveys related work both in systems as well as machine learning. Particularly, we cite related work on automated analysis of distributed systems and from machine learning, issues with dimensionality and transfer learning.

There has been a lot of research in the area of automated analysis of distributed systems in the past few years. Ours is one of the first to deal with scalability of such methods.

Two recent papers by Bodík *et al.* [7, 6] address issues with scale for analysis of problems at large internet sites using visualization and feedback from operators. Bodík *et al.* [7] proposed an automated statistical analysis tool along with a visualization tool to aid operators to detect and localize failures in a large-scale internet service based on user access patterns. In another work, Bodík *et al.* proposes visualization tools to aid operators in troubleshooting problems in large-scale internet services [6]. One tool provides a visual mapping of components and dependency relationships to make it easier to decipher the propagation of failures. It also allows operators to zoom in on "important" metrics for each component. In contrast, in our work, relevant metrics are automatically detected through the metric attribution and signature construction mechanisms. Another tool aids in the troubleshooting of recurrent problems by monitoring clickstreams of those operators who resolve the problems the first time. Our work uses searchable, indexable signatures, generated automatically, to retrieve similar occurrences along with operator annotations for previously resolved problems.

There have been additional work on performance diagnosis and debugging. Aguilera *et al.* describe two algorithms for isolating performance bottlenecks in distributed systems of opaque software components [2]. Their "con-

volution" algorithm employs statistical signal-processing techniques to infer causal message paths that transactions follow among components, which are not assumed to communicate via RPC-like request/reply pairs. At the opposite extreme of this knowledge-lean approach, Magpie characterizes transaction resource footprints in fine detail but requires that application logic be meticulously encoded in "event schema" [3]. The Pinpoint system of Chen *et al.* analyzes run-time execution paths of complex distributed applications to automatically detect failures by identifying statistically abnormal paths; faulty paths can then aid a human analyst in diagnosing the underlying cause [9]. Kiciman and Fox describe in greater detail the use of probabilistic context-free grammars to detect anomalous paths in Pinpoint [19]. Our approach shares with Pinpoint the use of statistical techniques, but the instrumentation we require is more readily available and we seek to diagnose performance problems rather than faults.All these methods have so far largely ignored the scalability issue.

P2 [20] is a novel way of building distributed applications by expressing network-oriented functionality as continuous queries over program and network state. More recently, Singh *et al.* [22] proposed a logging and monitoring facility built on top of P2 that provides a concise and powerful to express operations necessary to monitor and locate faults in large distributed systems. However, this solution is limited to systems built using P2.

The scalability issue, besides being a computational problem, is intrinsically difficult because of the curse of dimensionality. Many methods have been suggested in the machine learning literature to deal with the curse of dimensionality. Generally, these methods reduce the dimensionality of the problem by either projecting the metric space to a lower dimensional space, such as PCA [18], ICA [1] and random projection [13], or by selecting a subset of the original features in some fashion [16].

Projection methods use a linear or non-linear projection of the metrics to a lower dimensional space. These methods work well on many machine learning problems, but the features in the lower dimensional space have no semantic meaning, making interpretation difficult for operators. Besides, they are computationally expensive (cubic or quadratic in the number of metrics), and requires central collection of all original metrics. The second approach involves selecting a subset of the metrics using some optimality criterion (feature selection) [16]. These methods preserve the semantic meaning of the metrics. In our work we use greedy search algorithms for selecting a small subset of most relevant metrics in constructing models. However, most feature selection methods are at least quadratic in the number of features and the number of samples. As we demonstrated, without partition of the metrics, these methods also become computationally expensive.

Finally, in this work we use transfer learning methods for leveraging similarities between different instances. To our knowledge, our work is the first to use transfer learning methods for diagnosing performance in computer systems. Most related to our work is [4], in which models for email virus detection are trained using a transfer learning method called Latent Dirichlet Allocation (LDA) [5]. Our method is a simpler form of transfer learning, feasible because it is applied to the replicated instances of the internet service with similar metrics.

6 Discussion and Future Work

In this section we discuss several issues and observations from our results and experiences with the daily operations of the real internet services. We also suggest future directions for our work stemming from those observations.

Transfer learning: The positive results using our transfer learning of models are very encouraging, but it is important to note that the transferring of models leverages the similarities between the instances, but does not necessarily account for direct dependencies between the instances that could aid in diagnosing a particular problem. For example, if a problem on instance x (e.g., overloaded CPU) causes performance problems on a neighboring instance y (due to the load balancer), the dependency of y on x is not captured by transferring of models. We can account for such dependencies by extending our method to learn models of instance y adding the metrics collected on instance x as another set of data sources (and metrics from any other instances for which there are known dependencies). Such a method is scalable when using our multi-source temporal segmentation. We have tested this method on our traces. However, we did not find that using metrics of dependent instances provided any benefits, even though in some cases there were meaningful models trained. In these cases, we observed that it was not necessary to use metrics from related instances, as the metrics of the instance already provided all the required diagnostic information. We believe that use of related instance metrics should thus be used selectively to produce models, perhaps with rules triggering the analysis of those metrics when no plausible problem signatures are produced by the metrics of the instance.

We also note that our transfer learning method can be used to quickly provide diagnostic capabilities to newly deployed service instances. This is very important virtual machines are used more frequently in data centers to dynamically add and remove capacity. A new instance, which lacks any historical data, can leverage the models learned on previous instances to quickly and correctly identify performance problems, which would otherwise require long trace collection. The success of transfer learning hinges on known similarities between the instances; we intend to investigate the limits of this transfer as more differences (hardware, configurations, etc.) are introduced.

Log processing: We showed the method for processing text event logs described in Section 2.4 was very efficient and accurate in processing our logs. We observed similar results on logs collected from very large data centers, with a wide variety of services and error messages. To the reader familiar with natural language processing, these positive results are perhaps surprising. First, our method takes into account the order of words in the message and does not allow for insertions or deletions of words; this can make very similar messages appear far with our distance measure. We have seen little sensitivity to this issue in our logs. Also, extending our method to account for these is easy and there are well established methods we intend to use. Second, in natural language, small changes in a sentence can have

very different meanings, e.g., "memory is not sufficient" vs. "memory is sufficient", or "database system is down" vs. "webserver system is down"; with our method, the two messages can appear near and get clustered together. However, cases of the first examples are less likely, as logs typically refer to errors. Cases such as the second example are more plausible (although we have not seen such cases in our logs). For this problem we can enhance our approach with some domain specific keywords (potentially supplied by the operators). Having different keywords prevents messages from being grouped together. We intend to investigate these extensions as part of our future work.

A second limitation of our approach is that we treat all error messages from the application as a *single* partition. Large applications can have a lot of unique error messages, which maps to a large metric space. This poses the scalability issue again. We plan to develop an automatic way of partitioning metrics from the same data source, if the data source contains too many metrics.

7 Conclusions

This paper presented a scalable approach for automated identification of probable causes of performance problems in large server systems with geographically replicated sites, multiple tiers, and multiple system instances per tier. We demonstrated scalable use of learning to automatically associate performance problems (identified by SLO violations) with the system, application, or log attributes that are most relevant to them. The resulting service yields possible explanations that greatly aid with system troubleshooting. It removes the need for manual inspection of large volumes of performance data in search of anomalies that might explain the performance problem.

We have demonstrated three major architectural improvements that lead to the scalability of our approach. First, our algorithms divide the space of analyzed metrics into bounded partitions that reduce learning overhead while preserving accuracy. The conflicting goals of low overhead and high accuracy were jointly achieved thanks to the multi-source temporal segmentation algorithm, run among the learners of these partitions. It was shown that by simply allowing one learner to inform others when a good model was found, learning accuracy could be significantly improved. This improvement is attributed to the fact that another learner (with only a poor model) could then know to give up and reset its window, hence preventing distortion of models when the metrics analyzed by the learner have no correlation with the problem observed. Further, by combining the best models from the set of learners a much superior ability to associate metrics with problems is achieved.

Second, we have illustrated the use of multiple qualitatively different metrics to potentially explain problems. In particular, in addition to system and application metrics, we have demonstrated the use and analysis of logs. Since logs often contain expressive human-readable messages, they can be particularly indicative of the nature of problems.

Finally, we demonstrated the use of transfer learning, whereby different learners exchange models of common problems. It was shown that indeed models learned in one server installation can help identify similar problems on another. Our techniques were evaluated using production clusters with multiple sites distributed across a wide area network. In addition to showing improvements in learning accuracy, retrieval quality, and scalability, we presented specific case studies that illustrate the prescriptive power of the approach in identifying practical problems in large-scale multi-instance real-life applications. We believe the proposed approach is the first learning-based systems contribution that allows automated diagnosis to scale well to the size of realistic enterprise applications.

References

[1] E. O. Aapo Hyvrinen, Juha Karhunen. *Independent Component Analysis*. Wiley-Interscience, 2001.

[2] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proc. 19th ACM SOSP*, 2003.

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

[4] M. Barreno, B. Nelson, R. Sears, and A. D. Joseph. User model transfer for email virus detection. In *First workshop on tackling computer system problems with machine learning techniques (SysML)*, 2006.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of machine Learning Research 3*, 2003.

[6] P. Bodík, A. Fox, M. I. Jordan, D. Patterson, A. Benerjee, R. Jagannathan, T. Su, S. Tenginakai, B. Turner, and J. Ingalls. Advanced tools for operators at amazon.com. In *The First Annual Workshop on Autonomic Computing*, Dublin, Ireland, 2006.

[7] P. Bodík, G. Friedman, L. Biewald, H. Levine, G. Candea, A. Fox, M. I. Jordan, D. Patterson, K. Patel, G. Tolle, and J. Hui. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *Proc. 2nd International Conference on Autonomic Computing (ICAC'05)*.

[8] G. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3, 1950.

[9] M. Chen, E. Kıcıman, E. Fratkin, E. Brewer, and A. Fox. Pinpoint: Problem determination in large, dynamic, Internet services. In *Proc. International Conference on Dependable Systems and Networks*, pages 595–604, Washington, DC, June 2002.

[10] I. Cohen and M. Goldszmidt. Properties and benefits of calibrated classifiers. In *ECML/PKDD*, 2004.

[11] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. S. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proc. 6th USENIX OSDI*, 2004.

[12] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox. Capturing, indexing, clustering, and retrieving system history. In *Proc. 20th ACM SOSP*, 2005.

[13] S. Dasgupta. Experiments with random projection. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 143–151, 2000.

[14] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, second edition, 2001.

[15] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29:131–163, 1997.

[16] I. Guyon and A. Elisseef. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 2003.

[17] L. J. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

[18] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[19] E. Kıcıman and A. Fox. Detecting application-level failures in component-based internet services. *IEEE Transactions on Neural Networks*, Spring 2005.

[20] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. In *20th SOSP*, 2005.

[21] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[22] A. Singh, T. Roscoe, P. Maniatis, and P. Druschel. Using queries for distributed monitoring and forensics. In *EuroSys 2006*.

[23] The Open Group. Application Response Measurement (ARM) 2.0 Technical Standard, July 1998. `http://www.opengroup.org/onlinepubs/009619299/toc.pdf`.

[24] S. Thrun and L. Pratt. *Learning to Learn*. Springer, 1998.

[25] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Academic Press, 2000.

[26] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. In *DSN*, 2005.