Anomaly Detection and Diagnosis in Grid Environments

Lingyun Yang[1] Chuang Liu[2] Jennifer M. Schopf [3] Ian Foster[1,3]

[1]Department of Computer Science, University of Chicago, Chicago, IL 60637

[2]Microsoft, Redmond, WA 98052

[3]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439

lyang@cs.uchicago.edu chuangl@microsoft.com [jms, foster]@mcs.anl.gov

ABSTRACT

Identifying and diagnosing anomalies in application behavior is critical to delivering reliable application-level performance. In this paper we introduce a strategy to detect anomalies and diagnose the possible reasons behind them. Our approach extends the traditional window-based strategy by using signal-processing techniques to filter out recurring, background fluctuations in resource behavior. In addition, we have developed a diagnosis technique that uses standard monitoring data to determine which related changes in behavior may cause anomalies. We evaluate our anomaly detection and diagnosis technique by applying it in three contexts when we insert anomalies into the system at random intervals. The experimental results show that our strategy detects up to 96% of anomalies while reducing the false positive rate by up to 90% compared to the traditional window average strategy. In addition, our strategy can diagnose the reason for the anomaly approximately 75% of the time.

1. INTRODUCTION

Troubleshooting distributed Grids is a growing area of concern. Collaborations are increasing in size, with many more resources available for use and few ways to track or even detect failures or performance faults. Anomalous behavior of applications occurs frequently, but current techniques can be error prone and result in high false positive rates. One common cause is predictable periodic background behavior: shared network links have a daily pattern in that they are usually busier during the daytime [4], the number of jobs has a 24-hour pattern of many daytime submissions and then nightly draining of the queues [9], and periodic system administration tasks can have finer-grained, even hourly, occurrences.

SC07 November 10-16, 2007, Reno, Nevada, USA
(c) 2007 ACM 978-1-59593-764-3/07/0011…$5.00

The performance degradation caused by the periodic resource usage patterns can be predicted but is, in general, unavoidable if the resource is to be used. Hence, we see two kinds of anomalies in the system: *true anomalies,* which are caused by a failure or unexpected event, and *periodic anomalies,* which for the most part should not be considered anomalous as they are part of the normal system behavior. Only true anomalies should cause alert flags and be diagnosed.

Periodic behavior may differ widely, even on resources with similar capabilities, so any approach should be resource- and period-agnostic for wide applicability. For example, Figure 1 shows the CPU load collected on a machine in the Planetlab testbed for one week with a daily periodic usage pattern likely caused by the working schedule of the people sharing this machine. In contrast, Figure 2 shows the CPU load trace from a shared Linux machine at the University of Chicago with a usage pattern with a period of about 30 minutes, likely caused by system maintenance tasks.

The periodic resource behaviors will cause variations on the performance of applications running on these resources, which in turn may cause a large number of false positives when detecting anomalies. To address this situation, we propose a new strategy that uses signal-processing techniques to take normal periodic behavior into account, which greatly reduces the number of false positives. We extend the traditional window average-based anomaly detection approach (Section 2). Instead of analyzing the raw performance data collected, we automatically detect and *filter out* the periodic patterns existing in the resource performance data (Section 2.1). By analyzing the processed resource performance data, our strategy can *detect and diagnose* application anomalous behaviors (Section 2.2). We evaluate the effectiveness of our approach by applying it to three Grid applications (Section 3). Our results demonstrate that our strategy is able to *detect up to 96%* percent of the anomalies in a system with a much *smaller false positive* rate than standard statistical approaches.

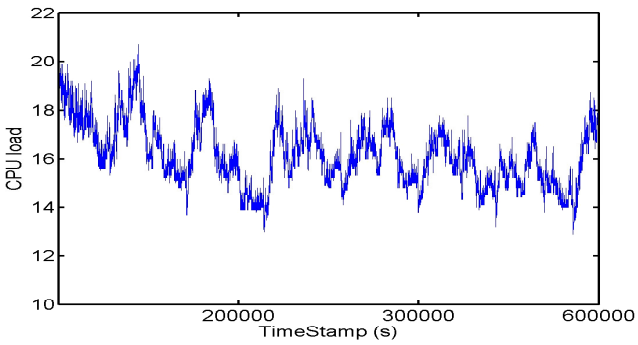

Figure 1: Daily periodic usage pattern of CPU load observed on a week-long trace for righthand.eecs.harvard.edu. Data was measured every 30 seconds.

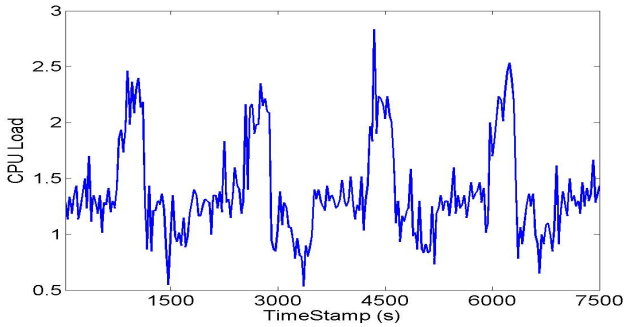

Figure 2: Typical two-hour CPU load trace from a Linux machine at the University of Chicago, vatos.cs.uchicago.edu, that shows a half-hour periodic variations. Data was measured every 30 seconds for two weeks during each monitoring period.

The main contributions of this paper are threefold:

1) A resource and period-agnostic approach to filtering out periodic behavior from trace data to better detect anomalies.

2) A diagnostic technique that is also resource independent and is more than 75% effective in determining the cause of anomalies.

3) Extensive experimentation showing these approaches are significantly better at detecting anomalies than the standard approach for CPU, memory-based, and network anomalies in both simulated and actual environments.

2. ANOMALY DECTION AND DIAGNOSIS

Window averaging is the most commonly used statistical anomaly detection method because of its simplicity and efficiency. It has been used to detect unexpected slowdown in the rate of wide-area file transfers [10], degradation in the throughput of distributed applications [3], and aberrant behaviors in network traffic [5]. This method maintains a moving window average across the performance data of interest as the baseline to compare against. If the current data is higher than the window average by some threshold value, it is tagged as an anomaly, and an alarm is sent. To achieve more stable results, one can modify this approach to send an alarm only after a minimum number of consecutive violations of the threshold [22].

We extend the use of the window average-based method by filtering the data using Fourier transforms to capture normal periodic behavior. We also add a diagnostic phase to the standard approach to understand the possible cause of the anomaly.

2.1 Filtering resource usage pattern using Fourier transform

Fourier transform-based spectral analysis is a natural choice for filtering periodic resource usage patterns because of its dominant capability in frequency domain analysis. Every sequence of performance measurements, called a *signal* in the signal-processing field, can be expressed in both a *time domain* and a *frequency domain* representation [23]. The time domain representation shows the amplitude of the measurements at successive time points, or how a signal changes over time. With only a time domain representation, however, it is difficult to answer questions about frequency behavior, such as "Does the data include any periodic signals?" and "What is the frequency and amplitude if there are any?" To answer these questions, we need to use the frequency domain representation of the signal, which shows how much of the signal's energy is presented as a function of frequency. We can transform between these time domain and frequency domain representations of the data by using Fourier transform and inverse Fourier transform.

Resource performance measurements are represented in the time domain. To identify periodic resource usage patterns, we Fourier transform the data to the frequency domain. After identifying and filtering out the periodic resource usage patterns in the data, we inverse Fourier transform the data back to the time domain representation. A similar approach has been used to detect periodic components in electrical signals [23] and photographs [12]. We remove all strong periodic signals by setting the amplitude of the corresponding periodic frequency to zero and then transform the resource performance data back to the time domain representation by applying an inverse Fourier transform to the frequency domain representation. The grey line and black line in Figure 3 show a before-and-after view of this process.

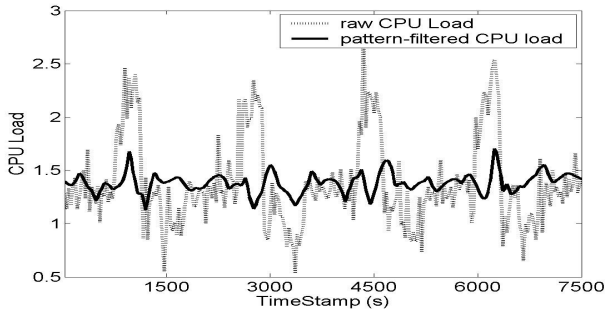

Figure 3: Raw CPU load and the CPU load after the pattern-filtering process.

The advantage of the Fourier transform method as a technique for identifying and removing periodic usage patterns in resource

performance data is that we do not need to know whether a periodic signal exists or what its frequency, amplitude, or shape is. The flexibility of the Fourier transform in dealing with periodic signals makes it possible to identify various periodic usage patterns in resource performance data automatically and dynamically.

2.2 Detection and diagnosis

We now introduce how to extend the traditional window average method in order to efficiently detect and diagnose anomalous application behaviors.

In our previous work [27], we proposed a statistical data reduction strategy for selecting related system metrics. The small subset of system metrics selected has been shown sufficient to capture the application behavior, thus reducing the data volume to be analyzed for anomaly detection and diagnosis. This data reduction strategy also builds a model between the application performance metric and selected system metrics, as follows:

$$Y=\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots\beta_n x_n, \qquad \text{Formula 1}$$

where Y is the application performance metric, x_i (i=1…n) are system metrics that describe resource performance and are selected by the data reduction strategy, and β_i (i=0…n) are a set of parameters obtained during the process of data reduction (please refer to [27] for the details of the model). Given this model, the parameters (the value of all β_i s), and measurements of system metrics (the value of all x_is), we can obtain an estimated application performance value(Y).

Using this model, we extend the traditional window average-based algorithm as follows:

1. Calculate a moving window average of the application performance data. This window average is used as the baseline for the application performance.
2. Compare the application performance data with the window average. If the current application performance value is within a threshold value of the average, this value is normal; go to step 6. (Section 3.1.1 discusses setting this parameter.)
3. To define whether the value is either a periodic anomaly or a true anomaly, first filter out the periodic patterns using Fourier transform techniques (Section 2.1). Then calculate an application performance value using Formula 1 with the filtered resource performance data. This value is the estimated application performance without the influence of the periodic resource usage patterns, which we refer to as the estimated application performance.
4. Compare the estimated application performance value with the window average. If the difference is more than the threshold value, this is an anomaly, so continue to step 5 for further diagnosis. Otherwise, the variation in the original application performance data is caused by periodic resource usage patterns, so this is not an anomaly. We then go to step 6 to exam the next application performance value.
5. Calculate the window average for each system metric, and use this window average as the baseline for resource behavior. When a real application anomaly is detected, check whether there is a large variation in the resource performance by comparing the current resource performance data with its window average baseline for each system metric, as done for the application performance. If there is a large performance variation during the same time period, the resource may be one of the causes of the anomalous application behavior.
6. Check the data at the next time point, and update the moving window average of the application performance data and resource performance data. Go back to step 2.

3. EXPERIMENTAL RESULTS

To evaluate the validity of our anomaly detection and diagnosis strategy, we applied it in three contexts: a Cactus application running in a shared local area network environment, a GridFTP transferring data across a simulated wide area environment, and a Sweep3d application running in a simulated wide area environment. We tested our methods to see whether they were able to detect anomalies that we introduced deliberately. In keeping with accepted practice [17, 29], we view our technique as effective if it can detect most of the anomalies (more than 90% in our experiments) and significantly reduce the false positives caused by periodic patterns of the resources.

3.1 Parameters

3.1.1 Window Average Threshold Value

If the difference between the performance data and the window average is more than the threshold value, a potential anomaly exists in steps 2 and 4 of our algorithm. Different applications may have different needs when defining what should count as an anomaly, which in turn will affect the setting of the threshold value. For our experiments, we set the threshold value to two standard deviations of the average application performance value. If the data is a normal distribution, a range around the mean plus or minus two standard deviations captures approximately 95% of the values, so a value outside this threshold has only a 5% chance of being normal behavior.

3.1.2 Window Size and Data Reduction Parameter

The two additional parameters that influence our anomaly detection results are the window size and the threshold parameter in the data reduction strategy, which determines how many system metrics will be selected and analyzed in the anomaly detection and diagnosis process. We selected these two values by running a set of experiments to search the space of feasible values on a training set of data collected by running the Cactus application on a shared cluster with 100 anomalies at random times.

We evaluated the quality of our anomaly detection approach using two criteria: the number of successful detections, HIT, and the number of false positives, FP. The parameter values that achieved the best anomaly detection results were selected for further evaluation. We first examined the sensitivity of the anomaly detection strategy on a selection of window sizes. We fixed the threshold parameter of the data reduction strategy equal to 0.95, as suggested by [27], and ran this strategy with different window sizes. Figure 4 shows the comparison of detection results with different window sizes. When the window size is small, the calculated window average fluctuates widely, and our strategy produces a higher number of false positives and lower hit rates. As the window size increases, the number of false positives decreases, and the hit rate increases. In this experiment, when the window size is larger than 32, the results flatten out, and our

strategy results in similar anomaly detection quality. When the window size is equal to 128, our strategy achieves the fewest false positives (53) and achieves a hit rate as high as 96%. We selected the window size equal to 128 for further evaluation.

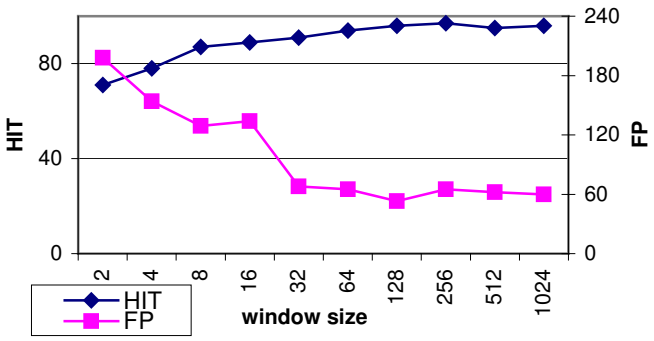

Figure 4: Anomaly detection results for different window sizes. The x-axis is in log scale, HIT is the number of detected anomalies (out of 100), and FP is the number of false positives.

We then examined the sensitivity of our anomaly detection strategy to the data reduction threshold parameter. We fixed the window size at 128 and ran the anomaly detection strategy with different threshold parameters, between 0 and 1 at intervals of 0.05, then calculated the number of hits and false positives achieved by our strategy for each value. The results are shown in Figure 5. As the data reduction threshold increases, more system metrics are selected, and more information is available to our anomaly detection strategy, thus allowing the anomaly detection strategy to achieve a higher hit rate. Our anomaly detection strategy achieves its highest hit rate (97%) when the data reduction threshold is equal to 0.90. The number of false positives also increases when more system metrics are used, but flattens quickly after the threshold value is larger than 0.3. We selected a threshold value of 0.90 for further evaluation because our strategy achieves the highest hit rate (97%) and produces 44 false positives, significantly less than the 676 produced by the traditional window average method on the same data.

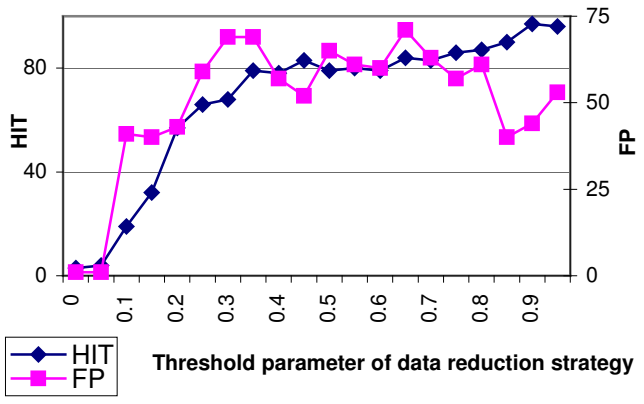

Figure 5: Anomaly detection with different data reduction threshold parameter for a window size of 128.

From these two experiments, we see that for both parameters, the detection result of our strategy fluctuates when the parameter values are small but flattens quickly after the two parameters reach certain values (32 for window size and 0.3 for data reduction parameter in our experiment) and stabilizes afterwards. We conclude that the values of these two parameters are not sensitive to our anomaly detection strategy as long as small values are avoided. Although we determine these parameters using the Cactus data, we used the same values for other application data in the following experiments. It would be possible to tune these parameters for each data set, and doing so might improve the performance of the anomaly detection algorithm, but it would greatly increase the overhead.

3.2 Cactus

We first tested our strategy on the Cactus application running in a cluster environment. Cactus [2] is a simulation of a 3D scalar field produced by two orbiting astrophysical sources. This application decomposes the 3D scalar field over processors and places an overlap region on each processor. We defined Cactus performance as the elapsed time per iteration.

3.2.1 Experimental Methodology

We ran Cactus on four shared Linux machines at the University of Chicago and collected measurements of system metrics selected by the data reduction strategy [27] and application performance measurements every 30 seconds. We collected four sets of Cactus data, each 2 weeks long. The first data set is used as training data to determine parameters in our strategy (Section 3.1.2). The other three data sets are used for verification purposes. We knew a priori that the resources experienced periodic variations, as shown in Figure 2.

For each of the four data sets, we manually inserted 100 anomalies during the execution of Cactus application by running resource consumption tools on the four machines at random times. The resource consumption tools were three simple programs that consume a given amount of CPU, memory, or bandwidth. Specifically, the CPU consumption tool ran several computation-intensive processes to compete for CPU time with the application. On a round-robin scheduled system (like a Linux system), if the resource consumption tool runs N ($\geq$10 in our experiment) processes, the application will get only 1/(N+1) percent CPU time; the other N/(N+1) percent of CPU time is used by the N computation-intensive processes. The memory consumption tool ran a process that allocated a set amount of memory space. The network consumption tool started multiple data transfers using the Linux scp command to transfer a large file to another computer to decrease the bandwidth available to our focus application.

3.2.2 Detection

We applied our strategy to three sets of Cactus data. To show the effectiveness of our strategy, we compare our modified window average method (Modified) to the traditional window average method (Traditional) using two metrics: the number of anomalies successfully detected and the number of false positives. As shown in Table 1, the traditional window average method produced about 600 false positives. Among them, about 90% are caused by the half-hour periodic variations in the resources performance. Our strategy, which considered the periodic resource usage pattern, can distinguish between true anomalies and periodic anomalies and eliminated between 84% and 91% of the false positives

produced by the traditional window average method, while still identifying between 93% and 96% of the injected anomalies.

Table 1: Anomaly detection results on Cactus data.

Data Set	Traditional		Modified	
	# of Hits	# of FPs	# of Hits	# of FPs
Cactus_Data 1	99	588	96	63
Cactus_Data 2	99	633	93	59
Cactus_Data 3	98	551	94	89

Our strategy extends the traditional window average method by filtering the periodic resource usage pattern for resource performance data. However, some anomalous variations in the resource performance measurement may contain components with frequencies similar to the frequencies of periodic patterns. These are also removed by mistake when we try to filter periodic patterns. Thus, these anomalies are weakened and cannot be detected by our strategy, resulting in a slightly lower hit rate than that of the traditional window method.

3.2.3 Diagnosis

To simplify the diagnosis process, we classified the system metrics that describe resource performance into three categories, CPU related, memory related, and network related, with a total of 12 possible causes across the four machines. Because the anomalies are introduced at random times, some anomalies may happen simultaneously or consecutively. Since our strategy detects the start and end of an anomaly by checking whether application performance exceeds the window average by some threshold value, if multiple anomalies overlap in time, only one alarm will be sent, but all possible reasons will be reported.

Table 2 shows the results of our analysis. For every detected anomaly, our strategy uses the system metrics selected by the data reduction strategy to give a diagnosis analysis and report possible reasons. We verify the diagnosis results by comparing the reasons reported (CPU related, memory related, or network related) with the type of anomaly inserted. These results show that, among the 93 to 96 anomalies detected on three data sets, our strategy was able to report the reasons for 82 to 87 anomalies correctly.

Table 2: Anomaly diagnosis results on Cactus data.

Data Set	# of Anomalies Detected	# of Anomalies Diagnosed Correctly
Cactus_Data 1	96	87
Cactus_Data 2	93	84
Cactus_Data 3	94	82

3.3 GridFTP data transfer

We test the capability of our strategy to detect networking anomalies using the Globus GridFTP data transfer tool on Emulab.

3.3.1 Experimental Methodology

To have a better understanding about the performance of our method on network anomalies, we need to control the introduction of anomalies and network topology during the experiment. Therefore, we ran GridFTP on the Emulab [1, 25] testbed. Emulab is an integrated experimental environment for distributed systems and networks. It provides a time-and space-shared platform for research and development by leveraging nodes allocated from resource pool and temporarily dedicated to individual users for emulation. Researchers access these resources by specifying a virtual topology. Users can modify the shape of the traffic of each link by changing the bandwidth, delay, and packets loss rate between any two nodes in the topology dynamically.

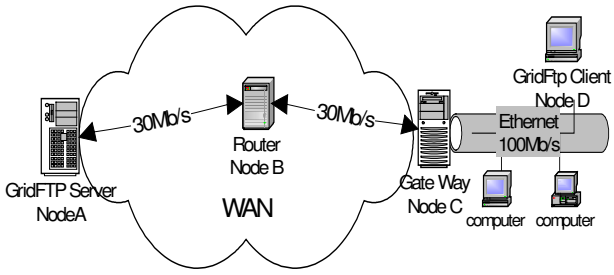

Figure 6: Emulated network topology in Emulab testbed.

We constructed a virtual topology in Emulab as shown in Figure 6. In this emulated distributed environment, machines in a LAN are connected with each other by 100 Mb/S Ethernet. They access the Internet through a gateway, denoted as Node C for convenience. A GridFTP server, denoted as Node A, is accessible by the gateway via s 30 Mb/s network. For complexity, we added a router, denoted as Node B, in the path from the GridFTP server to the gateway. If a client machine, denoted as Node D, in the LAN requests to transfer data from the GridFTP server, the data needs to pass three links. It is first sent from the GridFTP server to the router, via the link AB. The router then transmits the data to the gateway, via link BC. Finally, the data arrives at the client machine, via the link CD.

We ran GridFTP in this emulated distributed network and collected system metrics selected by our data reduction strategy and the GridFTP performance metric once every 30 seconds. Although there are four nodes on the path of data transfer, normally users will not (or are not allowed to) run monitors on the router and gateway. So we collect resource performance data only on the GridFTP server and client machines and the ping measurements from the client and server node to other three nodes, respectively. The performance metric for the GridFTP transfer is the data transfer rate, in megabits per second.

We collected three sets of GridFTP data. Each data set is about two weeks long. For each data set, we inserted 100 anomalies across the three links in the path between the client machine to the GridFTP server during the GridFTP data transfer by changing the traffic shaping parameters of each link in a random order. Emulab emulates the change of network traffic by a control network, which is invisible to applications, enforcing delay and bandwidth limitation of a network link. We introduced the anomalies into network links by changing the network configuration in the simulated network environment. For each anomaly, we decreased the bandwidth to a value less than 10% of its original value or increased the delay (or loss ratio) by 5 to 10 times of original value to cause significant performance slowdown in the GridFTP transfer rate.

We also tried inserting CPU and memory anomalies for GridFTP on Emulab testbed by introducing high CPU and memory load using the resource consumption tools as we did for Cactus

application. However, the results show that even very high CPU load and memory load (e.g., CPU load increases of more than 100 times) have no effect on the performance of GridFTP. One possible reason is that the GridFTP is implemented efficiently: it can scale to 1500 concurrent connections or more, so bandwidth is the only bottleneck.

3.3.2 Detection

To show the effectiveness of our strategy, we applied our modified window average method (Modified) and the traditional window average method (Traditional) to the three sets of GridFTP data and compared the results of these two strategies as we did for Cactus application. We used the same two metrics, namely, the number of hits and number of false positives, to evaluate the two strategies. Because Emulab emulates the network topology by allocating physical nodes temporally dedicated to a user, there is no usage pattern caused by resource sharing or periodic system maintenance jobs in these resources. Hence, the traditional window average method produces only several false positives caused by noise. Our strategy reduces the number of false positives to 2 to 7 and can still detect more than 90 anomalies successfully, as shown in Table 3.

Table 3: Anomaly detection results on GridFTP data.

Data Set	Traditional		Modified	
	# of Hits	# of FPs	# of Hits	# of FPs
GridFTP_Data 1	99	5	92	2
GridFTP_Data 2	97	9	95	7
GridFTP_Data 3	100	6	90	4

From this result, we can see that our method is not as efficient as the traditional method for detecting anomalies when there is no periodic usage pattern in the resource performance. When preprocessing the resource performance measurements, we tried to remove the periodic patterns from the original performance data. Although doing so helps remove false positives caused by periodic patterns, it possibly incorrectly treats a very small fraction of anomalies as false positives. On actual networks we would expect to see various periodoc behaviors, so our results would likely improve.

3.3.3 Diagnosis

The anomalous network behaviors in the Emulab testbed are emulated network environment. Although applications running on the emulated network can "sense" anomalous network behaviors, these anomalous system behaviors are not really reflected in the low-level system metrics. To diagnose anomalous network behaviors, we used an application-level command, ping, to measure network behaviors. We show an example in Figure 7.

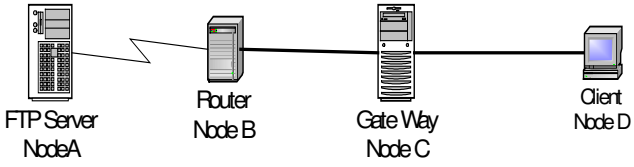

Figure 7: Network path from GridFTP server to client machine.

There is a delay increment in the link A-B, shown as a crooked line. Using ping measurements, we can determine this phenomenon because we will see a performance hit on the ping tests A_B, A-C, A-D and D-A, but not on D-B and D-C. In Table 4 we show the relation between network links and ping measurements between nodes. The table shows that for any anomalous link combination, except the last two, we can determine what is affected simply by using pings.

Table 4: Anomalous links and corresponding anomalous ping measurements.

Anomalous Links	Anomalous Ping Measurements
Link AB	A to B, A to C, A to D, D to A
Link BC	A to C, A to D, D to A, D to B
Link CD	A to D, D to A, D to B, D to C
Link AB and Link BC	A to B, A to C, A to D, D to A, D to B
Link BD and link CD	A to C, A to D, D to A, D to B , D to C
Link AB and Link CD	A to B, A to C, A to D, D to A, D to B, D to C
Link AB, Link BD, Link CD	A to B, A to C, A to D, D to A, D to B, D to C

The experimental results of our detection algorithm on the GridFTP data are shown in Table 5. For the 92 to 95 anomalies detected, our strategy finds the problematic links for 73 to 81 anomalies correctly. Remember we diagnose the application anomalies by relating the resource anomalous behaviors to the anomalous application behaviors. As discussed in Section 3.3.2, the signal-processing techniques will remove some resource anomalies information incorrectly as noise or periodic usage pattern. Moreover, some resource anomalous behavior cannot be detected. Thus we cannot find the proper reasons for the anomalous application behavior.

Table 5: Anomaly diagnosis results on GridFTP data.

Data SSet	# of Anomalies Detected	# of Anomalies Diagnosed
GridFTP_Data 1	92	73
GridFTP_Data 2	95	81
GridFTP_Data 3	90	74

3.4 Sweep3d

We used Sweep3d [16] to validate our strategy for applications running in Grid environment. Sweep3D is a 3D discrete ordinates neutron transport application that runs on multiple processors using domain decomposition and MPI message passing. The performance metric is the elapsed time per iteration. The execution of Sweep3d includes both network communications and computation. With varying problem size, the computation//communication ratio will change, so the application shifts from network-bound to CPU-bound.

3.4.1 Experimental Methodology

To have a better understanding about the relationship between different resource periodic patterns, application behaviors, and their effects on anomaly detection and diagnosis, we ran Sweep3d

on a simulated Grid environment using Emulab to control the introduction of resource periodic patterns and anomalies.

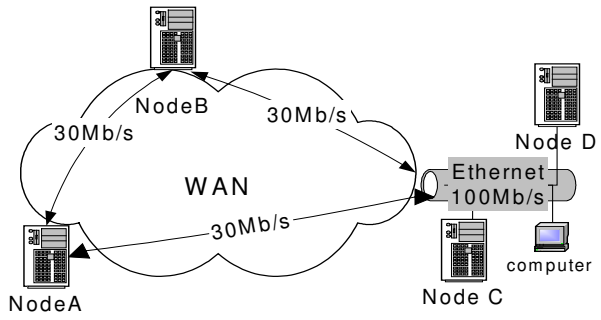

Figure 8: Emulated network topology in Emulab testbed for Sweep3d application.

We constructed a distributed virtual WAN environment in Emulab testbed, shown in Figure 8. In this emulated distributed environment, Sweep3d runs on four machines from three domains. Machines from different domains communicate via 30 Mb/s network links. Machines C and D belong to the same domain and are connected via 100 Mb/S Ethernet. We use the one-dimensional decomposition to partition the workload of Sweep3d application. Therefore, communication happens on three network links in this experiment: the link between machine A and B, the link between machine B and C, and the link between machine C and D.

For this experiment, we also emulated various periodic CPU load patterns for machines from different domains by running some CPU-intensive programs on these machines. Machine A has a daily periodic CPU load pattern, with amplitude equal to 5; machine B has a two-hourly periodic CPU load pattern, with amplitude equal to 3. There are no periodic patterns on machines C and D.

We ran Sweep3d in this emulated distributed environment and collected resource performance data selected by our data reduction strategy and the Sweep3d elapsed iteration time every 30 seconds. We collected resource performance data on all four computing machines and ping measurements from each pair of machines. To show how our anomaly detection strategy works on different application settings, we also varied the problem size of the Sweep3d application to change its computation/communication ratio and thus show different application behavior.

We chose three problem sizes to change the computation and communication ratio of the Sweep3d application. When the problem size is small, the computation/communication ratio is small. The communication is the application performance bottleneck, and periodic CPU usage patterns do not have a significant influence on the performance of the Sweep3d application. When the problem size increases, so does the computation /communication ratio. We collected three sets of Sweep3d data. Each data set includes performance data for three problem sizes for 9 days, with each problem size running for about 3 days.

For each data set, we inserted 100 anomalies into each of the three communication links by changing the traffic shaping parameters of each link in a random order. The performance data for the three problem sizes includes 33, 33, and 34 anomalies, respectively. We also tried introducing high memory load by using resource consumption tools as we had done for the Cactus application, but this did not affect the performance of the Sweep3d application because it is not memory bound for any problem size we tested.

3.5 Detection

To show the effectiveness of our strategy, we compared our modified window average method (Modified) to the traditional window average method (Traditional) for the three sets of Sweep3d data using the same two metrics, number of hits and number of false positives. The experimental results are shown in Table 6.

Table 6: Anomaly detection results for Sweep3d data.

Problem Size	Data Set	Traditional		Modified	
		# of Hits	# of FPs	# of Hits	# of FPs
Small	Sweep3d Data 1	33	1	31	0
	Sweep3d Data 2	33	4	29	3
	Sweep3d Data3	32	3	32	3
Medium	Sweep3d Data1	33	9	32	5
	Sweep3d Data 2	33	8	31	4
	Sweep3d Data 3	33	10	31	7
Large	Sweep3d Data 1	32	43	30	6
	Sweep3d Data 2	32	54	29	9
	Sweep3d Data 3	33	52	32	19

For the small problem size, there is no statistical difference between the two approaches, likely because the background CPU periodic behavior does not have a significant influence on the application performance, so there are few false positives to weed out. For the medium problem size, the traditional window average method produces several more false positives due to the daily periodic resource usage pattern. The Modified approach reduces about half of the false positives and misses 5 of the 99 anomalies, while the Traditional approach finds all of them; however, the Modified approach has a 40% reduced false positive rate. For the large problem size, the Traditional method produces about 50 false positives. Among them, approximately 90% are caused by the daily and 2 hourly periodic resource usage patterns. The Modified approach again misses some of the anomalies, this time 8 of 99, but has a significantly smaller false positive rate (as much as 7 times smaller) than that of the Traditional approach.

3.5.1 Diagnosis

Since the only anomalies of interest in this set of experiments were the result of network perturbations, we used logs of ping measurements in our diagnosis step to determine which network link was affected, as shown in Table 7. The experimental results show that, for the mixed Sweep3d application behaviors, our strategy diagnosed 85 to 89 anomalies on three sets of Sweep3d data, respectively.

Table 7: Diagnosis results on Sweep3d data.

Problem Size	Data Set	# of Anomalies Detected	# of Anomalies Diagnosed
Small	Sweep3d Data 1	31	30
	Sweep3d Data 2	29	28
	Sweep3d Data3	32	31
Medium	Sweep3d Data 1	32	31
	Sweep3d Data 2	31	28
	Sweep3d Data 3	31	27
Large	Sweep3d Data 1	30	28
	Sweep3d Data 2	29	29
	Sweep3d Data 3	32	31

4. RELATED WORK

Anomaly detection and diagnosis have been studied widely in many areas, including chemical processes management [8, 14, 30], materials control [26], mechanical fault detection [11, 19], and medical diagnosis [18]. In computer science, there is significant previous work for resource-level anomaly detection, especially network congestion detection [13, 24], and computer security management [7, 17, 20].

Anomaly detection and diagnosis in application-level performance often involve monitoring and analyzing the application or resource performance data and deducing the anomalous application behaviors. Allen et al. [3] detect performance contract violations using a window average-based method on the execution time of an application. Zhang et al. [29] show how to detect compliance with service-level objectives in a dynamic environment by managing an ensemble of Bayesian network models. Kelly [15] proposes using queuing theory observations together with standard optimization methods to yield accurate performance models to distinguish performance faults from mere overload. However, none of these approaches considers the influence of periodic resource usage patterns on application behavior.

Other anomaly detection work has considered periodic resource usage patterns but in a less flexible way. Burgess [6] proposes a memory-economic algorithm for detecting resource anomalies in event streams with either Poisson or long-tailed arrival processes using a pseudo-periodic function to address periodic variations in the performance data. Roughan et al. [21] present a simple and robust method that integrates routing and traffic data streams to detect forwarding anomalies using a number of models for anomaly detection, depending on the properties of the data in question. Both of these approaches require knowledge about the frequencies of the periodic patterns, if any, with a limited set of choices (generally only daily or weekly) and then select different models, formulas, or parameter values in the process of anomaly detection, instead of having a general approach, such as we do.

Zhang et al.[28] proposed using wavelet transforms to detect disease outbreaks (anomalies in this case) to filter seasonal periodicity before detection. However, the wavelet-based filtering method requires knowledge about the approximate range of the period and is has a coarser granularity than the Fourier transform-based method, which makes it inefficient when dealing with periodic signals whose frequencies do not fit the predetermined granularity of the wavelet decomposition well. Our strategy, using a Fourier-based method to filter periodic signal in the performance data, is much more flexible and efficient when dealing periodic signals with any frequencies and needs no knowledge of the periodicity a priori.

5. SUMMARY AND FUTURE WORK

Periodic variations in resource performance is normal and inevitable and can cause a high false positive rate when doing anomaly detection with standard approaches. In this paper we present an approach to anomaly detection and diagnosis strategy that extends traditional methods by using signal processing techniques to filter out periodic resource variation, regardless of the type of resource or period. In addition, we develop a diagnosis technique to determine which resource is the probable cause of an anomaly.

Independent of the periodic resource usage patterns, applications, and network configurations, our experimental results show that our strategy detects up to 96% of anomalies while reducing the false positive rate up to 90% when compared to the traditional window average strategy. In addition, our strategy can diagnose about 70% to 90% of reasons correctly.

Our anomaly detection and diagnosis strategy uses a window average-based method to detect anomalies. We plan to study other anomaly detection methods such as artificial neural networks methods and hidden Markov model methods, and compare them with window-based methods. We argue that our idea of considering periodical usage patterns when detecting application anomalies is also applicable to these methods. Indeed, the techniques that we have described here can be used as a complement to these advanced anomaly detection techniques to de-noise and filter periodically usage patterns before we apply these anomaly detection methods on resource performance data. In this way, we could reduce the false alarms caused by noise and by periodical resource usage patterns.

ACKNOWLEDGEMENTS

This work was supported in part by the U.S. Department of Energy under Contract DE-AC02-06CH11357.

REFERENCES

[1] "Emulab Tutorial: http://www.emulab.net/tutorial/docwrapper.php3?docname=tutorial.html."

[2] G. Allen, W. Benger, T. Goodale, et al., "The Cactus Code: A Problem Solving Environment for the Grid," 9th IEEE International Symposium on High Performance Distributed Computing (HPDC9), 2000.

[3] G. Allen, D. Angulo, I. Foster, et al., "The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment," University of Chicago, Chicago TR-2001-28, 2001.

[4] P. Barford, J. Kline, D. Plonka, et al., "A Signal Analysis of Network Traffic Anomalies," Proceedings of ACM SIGCOMM Internet Measurement Workshop, 2002.

[5] J. D. Brutlag, "Aberrant Behavior Detection in Time Series for Network Monitoring," Proceedsing of the 14th Systems Administration Conference, 2000.

[6] M. Burgess, "Probabilistic Anomaly Detection in Distributed Computer Networks," *Science of Computer Programming*, vol. 60, pp. 1-26, 2006.
[7] K. Das, "Protocol Anomaly Detection for Network-based Intrusion Detection," 2001.
[8] S. Dash, R. Rengaswamy, and V. Venkatasubramanian, "Fuzzy-logic based trend classification for fault diagnosis of chemical processes," *Computers and Chemical Engineering*, pp. 347-362, 2002.
[9] A. B. Downey, "A Parallel Workload Model and its Implications for Processor Allocation," *Cluster Computing*, vol. 1, pp. 133-145, 1998.
[10] D. Gunter, M. Rodriguez, B. Tierney, et al., "Dynamic Anomaly Detection of a Wide Area File Transfer Service," Submitted to SC06, 2006.
[11] J. M. House, W. Y. Lee, and D. R. Shin, "Classification Techniques for Fault Detection and Diagnosis of an Air-Handling Unit," *ASHRAE Transactions*, vol. 105, pp. 1987-1997, 1999.
[12] A. Igor, B. Constantine, D. E. R, et al., "Frequency domain median-like filter for periodic and quasi-periodic noise removal," *International Society for Optical Engineering Proceedings Series*, 2002.
[13] V. Jacobson and M. J. Karel, "Congestion Avoidance and Control," Proceedings of the SIGCOMM '88 Symposium, 1988.
[14] M. Kano, K. Nagao, S. Hasebe, et al., "Comparison of Statistical Process Monitoring Methods: Application to the Eastman Challenge Problem," *Computer and Chemical Engineering*, vol. 24, pp. 175-181, 2000.
[15] T. Kelly, "Detecting Performance Anomalies in Global Applications," Second USENIX Workshop on Real, Large Distributed Systems (WORLDS 2005), 2005.
[16] K. R. Koch, R.S.Baker, and R. E. Alcouffe, "Solution of the First-order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor," *Trans. Amer. Nuc. Soc.*, vol. 65, 1992.
[17] M. V. Mahoney, "Network Traffic Anomaly Detection Based on Packet Bytes," Proc. ACM-SAC, 2003.
[18] A. S. Minhas and M. R. Reddy, "Neural Network Based Approach for Anomaly Detection in Lungs Region by Electrical Impedance Tomography," *Physiological Measurement*, vol. 26, pp. 489-502, 2005.
[19] A. G. Parlos, K. Kim, and R. Bharadwaj, "Sensorless Detection of Mechanical Faults in Electromechanical Systems," *Mechatronics*, vol. 13, pp. 357-380, 2004.
[20] A. G. Pennington, J. D. Strunk, J. L. Griffin, et al., "Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior," *12th USENIX Security Symposium*, 2002.
[21] M. Roughan, T. Griffin, Z. M. Mao, et al., "IP Forwarding Anomalies and Improving Their Detection Using Multiple Data Sources," Proceedings of the ACM SIGCOMM workshop on Network Troubleshooting: research, theory and operation practice meet malfunctioning reality, 2004.
[22] V. A. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting SYN flooding attacks," Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE, 2004.
[23] S. W. Smith, *The Scientist and Engineer's Guild to Digital Signal Processing*. San Diego, California: California Technical Publishing, 1999.
[24] M. Welzi, *Network Congestion Control: Managing Internet Traffic*: Wiley, 2005.
[25] B. White, J. Lepreau, L. Stoller, et al., "An Integrated Experimental Environment for Distributed Systems and Networks," 5th Symposium on Operating Systems Design and Implementation (OSDI), 2002.
[26] R. Whiteson, F. Kelso, C. Baumgart, et al., "An Anomaly Detector Applied to a Materials Control and Accounting System," 35th Annual Meeting of the Institute of Nuclear Materials Management, 1994.
[27] L. Yang, J. M. Schopf, C. L. Dumitrescu, et al., "Statistical Data Reduction for Efficient Application Performance Monitoring," CCGrid 2006, 2006.
[28] J. Zhang and F. C. Tsui, "Detection of Outbreaks from Time Series Data Using Wavelet Transform," AMIA Annu Symp Proceeding, 2003.
[29] S. Zhang, I. Cohen, M. Goldszmidt, et al., "Ensembles of Models for Automated Diagnosis of System Performance Problems," IEEE Conference on Dependable Systems and Networks (DSN), 2005.
[30] Y. Zhou, J. Hahn, and M. S. Mannan, "Fault Detection and Classification in Chemical Processes Based on Neural Networks with Feature Extraction," *ISA Transaction*, vol. 42, pp. 651-664, 2003.