Identifying Propagated Response Delays in Performance Monitoring of n-Tier Applications

Yasuhiko Kanemasa, Atsushi Kubota, Hirokazu Iwakura,
Junichi Higuchi, Yuji Nomura
System Software Laboratories
FUJITSU LABORATORIES LTD.
Kawasaki, Japan
Email: {kanemasa, kubota.atsushi, iwakura,
higuchi.junichi, nomura.yuji}@jp.fujitsu.com

Toshinori Arai, Susumu Nakadate, Hiroshi Kanou
Network Security Services Division
FUJITSU FSAS INC.
Kawasaki, Japan
Email: {arai.toshinori, nakadate,
kanou.hiroshi}@jp.fujitsu.com

***Abstract*—One of the significant challenges on performance monitoring of an n-tier system is the "response delay propagation", in which a response delay in a component server is propagated to other component servers due to the invoking relations among request types in different component servers of the system. It leads the operations manager of the system to misdiagnose the location of source delays and results in wasting time to investigate the root cause. We developed a response delay monitoring system that helps the operations managers distinguish the source delays from many other propagated delays. The system is able to build a model of invoking relations among request types in different component servers and use the model to diagnose the response delay propagation and pin-point the location of source response delays. To obtain such invoking relations among request-types from black-box component servers in an n-tier system, we propose a novel invoking relation estimation method which can achieve high accuracy of true invoking relation among request types by eliminating the negative influence of two spurious correlation factors through partial correlation analysis. We implemented the response delay monitoring system and evaluated the effectiveness of our invoking relation estimation method on a real in-company n-tier system which has thousands of request-types in each tier. The result (over 90% in precision) confirms our estimation method can effectively capture invoking relations in an n-tier system.**

***Keywords*—N-tier system; Performance diagnosis; Response delay; Correlation analysis; Invoking relation; Operations manager**

I. Introduction

In datacenters, operation managers continuously monitor the QoS (quality of service) of enterprise applications to keep good QoS of them. In many cases, such enterprise applications consist of multiple component servers, each of which has a different role. A typical one is a web-facing three-tier system which consists of web servers to process end-users' HTTP requests, application servers to process computational workload generated by the requests, and database servers to keep consistency of the data for the application. Since the multiple tiers cooperate with each other to process each client request, it is challenging for an operations manager to correctly diagnose the root cause when the system encounters unexpected performance.

One significant challenge of monitoring an n-tier system is the propagation of response delays among tiers (Fig. 1). Through invoking relations among request-types in different tiers, a response delay of a request-type in a downstream tier causes a response delay of its invoker request-type in an upstream tier since the processing of the request-type in the upstream tier needs to wait a reply from the downstream tier. To distinguish the two types of response delays (the source ones and the propagated ones) is not easy when the number of request-types becomes large. For example, in the case of our in-company n-tier system used in the evaluation, there are a few thousand request-types in the system (see Section IV-A). In this case, how to effectively distinguish the two types of response delays becomes important in order to quickly identify the root cause of performance degradation.

If we can obtain the invoking relations among request-types in an n-tier system, it is possible to precisely diagnose response delay propagations among request-types and distinguish the source delays from the propagated delays. In real, however, it is difficult to obtain such invoking relations since they are dynamically determined in the source code of the application, reading and understanding a large amount source code is a time-consuming task, and further many applications disallow revealing their source code to a third party. An alternative way to obtain the invoking relations among request-types is to estimate them through black-box observations from outside of

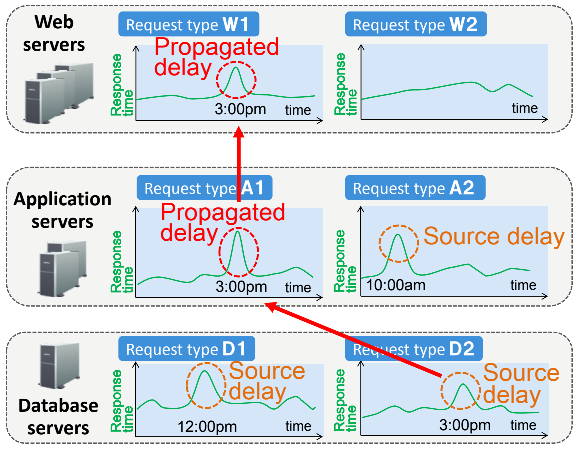


Fig. 1: Response delay propagation in an n-tier system

the target system. For instance, Hu *et al.* [9] proposed a method to estimate invoking relations among VMs in a virtualized datacenter by using a correlation analysis on the statistic of their networking.

However, there are two technical problems which decrease the accuracy of such an estimation if we use the simple correlation analysis technique for collecting invoking relations among request-types in an n-tier system. First, the request-frequency of each request-type is correlated with the total number of client accesses to the system. Thus, even though two request-types have no invoking relation, they may have a certain (spurious) correlation because they share a similar frequency changing trend (the total number of client accesses). Second, there is a case that some multiple request-types are frequently invoked simultaneously by a client. Concretely, a web browser issues HTTP requests for multiple URLs simultaneously to obtain one web page which consists of multiple frames. In that case, such a simultaneous invoking of multiple request-types makes a spurious correlation when we apply a correlation analysis on their request-frequency, and decreases the accuracy of invoking relation estimation.

The main contribution of the paper is an accurate estimation technique of invoking relations between two request-types executed in different tiers in an n-tier system, based on a correlation analysis between the request-frequency time-series of one request-type and that of the other request-type (Section II). The technique adopts partial correlation analysis to eliminate the influences of the above two spurious correlation factors, and achieves a high accuracy on the estimation. Our experimental evaluations with a real large[1] in-company n-tier system show the precision of estimation is 91.0%.

As the second contribution of the paper, we developed a response delay monitoring system for an n-tier system which monitors response delays of each request-types and visualize the propagations among them (Section III). It can reduce redundant alerts for response delays caused as the result of response delay propagations from downstream tiers, by diagnosing such delay propagations using the invoking relations pre-generated by our estimation technique.

The remainder of this paper is structured as follows. Section II introduces the highly accurate estimation technique of invoking relations between request-types. The overall response delay monitoring system which uses the obtained invoking relations for delay propagation diagnosis is presented in Section III. In Section IV, experimental evaluations with a real in-company two-tier system show how accurately our invoking relation estimation method can extract invoking relations between request-types. Related works are summarized in Section V, and Section VI concludes the paper.

II. Estimation of Invoking Relations

In this section, we show our novel estimation method of invoking relations among request-types in every two consecutive tiers of an n-tier system. The obtained invoking relations are used for diagnosing response delay propagation as we will present in Section III.

[1]It means "large" in function number wise and source code size wise. The large number of functions (menus) in the system generate such a large number of request-types.

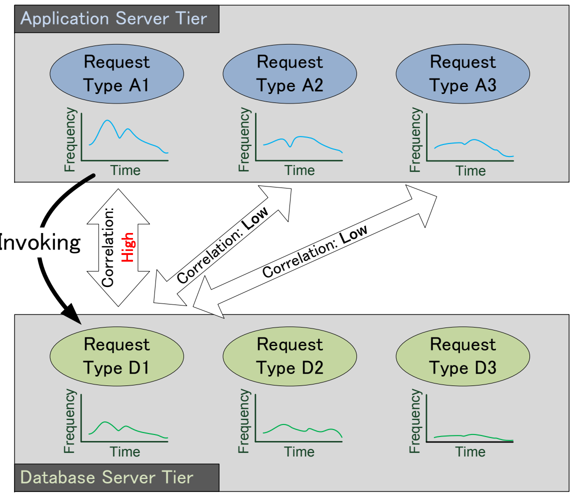


Fig. 2: Estimation of an invoking relation between request-types by applying correlation analysis on their request-frequency time-series.

Correlation analysis is a typical way to extract a hidden relation between two sets of samples. It is further used to estimate the hidden relationship among component servers inside a distributed system [9]. We adopted the same approach to estimate invoking relations, which applies a correlation analysis between each pair of request-types, each of which is from a different tier in an n-tier system. In the case of an illustrative example in Fig. 2, the request-type D1 in the database server tier is estimated as invoked by the request-type A1 in the upstream application server tier since the frequency of the two request-types in continuous time windows show high correlation. This methodology is based on an assumption that the request-frequency time-series data of two request-types which are in an invoking relation should have high correlation while that of two request-types without an invoking relation should show very low correlation

However, in practice, the request-frequency time-series data of two request-types which are not in an invoking relation frequently still show a certain amount of correlation due to two factors that we will show in Section II-A. Such spurious correlation leads to many false positives in process of estimating invoking relations among request-types from different tiers resulting in low accuracy on the estimation. The Section II-B and Section II-C show how our invoking relation estimation method eliminates the two spurious correlation factors, respectively. Finally, how to test the correlation is presented in Section II-D.

A. Two Problems of Applying Correlation Analysis for Invoking Relation Estimation

The first factor of spurious correlation is the influence of overall request-frequency trend. Since the overall request-frequency of all request-types in a tier has a time trend (e.g. the overall request-frequency shows a steep rise at the start of a business day) and the request-frequency of every request-type in each tier follows the same time trend, so it is natural that

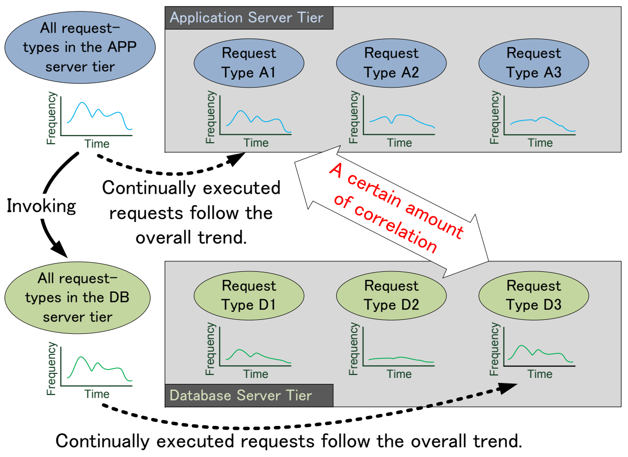

Application Server Tier
All request-types in the APP server tier
Request Type A1
Request Type A2
Request Type A3
Frequency
Time
Invoking
Continually executed requests follow the overall trend.
A certain amount of correlation
All request-types in the DB server tier
Request Type D1
Request Type D2
Request Type D3
Database Server Tier
Continually executed requests follow the overall trend.

Fig. 3: Spurious correlation caused by the influence of the overall request-frequency time trend.

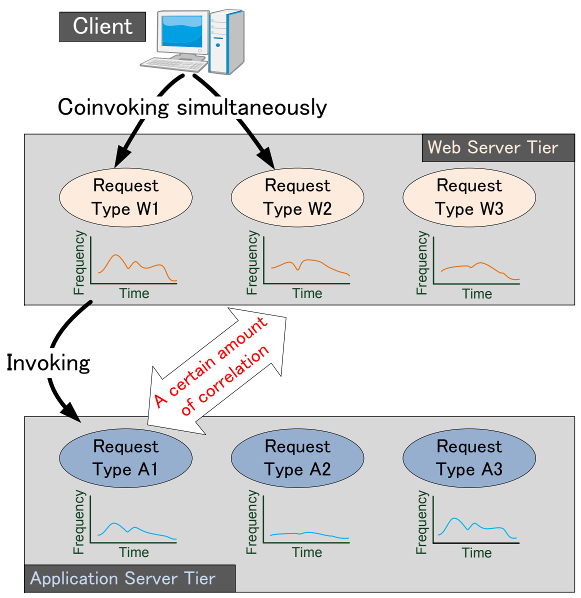


Fig. 4: Spurious correlation caused by the influence of coinvoked request-types.

two request-types from two different tiers follow the overall time trend and thus have certain correlation between them even though they have no invoking relation (see Fig. 3). We call this type of correlation as spurious correlation since there is no real invoking relation between the two request-types.

The second factor of spurious correlation is the existence of coinvoked request-types. In the case of request-types in the top most tier in an n-tier system, it happens that two request-types are frequently invoked simultaneously by a client. For example, two different HTTP request-types can be coinvoked by a client (see Fig. 4) because each of them is invoked from the respective frame in one web page. In such a case, a spurious correlation happens between a request-type W2 in an upstream tier and a request-type A1 in its adjacent downstream tier invoked from a request-type W1 in the upstream tier which is coinvoked with W2 from clients since their request-frequency time-series obviously have a certain amount of correlation.

B. *Eliminating the Influence of Overall Time Trend*

We adopt partial correlation analysis to eliminate the influence on correlation analysis caused by the request-frequency trend of all request-types in a tier. The key concept in partial correlation analysis is the partial correlation coefficient $r_{xy.z}$ between variables x and y, eliminating the influence of a third variable z. The partial correlation coefficient $r_{xy.z}$ can be calculated from the pairwise values of the correlation between variables x , y , and z (r_{xy}, r_{yz}, r_{xz}) as follows:

$$r_{xy.z} = \frac{r_{xy} - r_{xz}r_{yz}}{\sqrt{1-{r_{xz}}^2}\sqrt{1-{r_{yz}}^2}} \tag{1}$$

For example, let variables x, y and z be 8-element vectors as follows:

$$\begin{aligned} x &= (1,2,5,6,7,1,5,3), \\ y &= (3,4,8,15,14,8,10,6), \\ z &= (11,19,53,69,56,36,49,31). \end{aligned}$$

Here, correlation coefficients r_{xy}, r_{yz}, r_{xz} are 0.853, 0.937, 0.873, respectively. It seems like variable x and variable y are highly correlated judging from the high correlation coefficient value 0.853 between them. However, after eliminating the influence of variable z using (1), the partial correlation coefficient between x and y becomes 0.205 as follows, which is no longer high correlation:

$$r_{xy.z} = \frac{0.853 - 0.937 * 0.873}{\sqrt{1-0.937^2}\sqrt{1-0.873^2}} = 0.205$$

Considering the correlation coefficient between the request-frequency time-series data of a request-type U1 in an upstream tier U (u_1) and that of a request-type D1 in a downstream tier D (d_1), there are two factors which influence the correlation coefficient and thus necessary to be eliminated; the request-frequency time-series of all request-types in the tier U (u_{all}) and that in the tier D (d_{all}). Further, the overall trend u_{all} influences the overall trend d_{all} since all of the request-types in the tier D are invoked by request-types in the tier U. Therefore, we first need to eliminate the influence of the overall trend u_{all} using correlation coefficients $(r_{u_1d_1}, r_{u_1d_{all}}, r_{d_1d_{all}}, r_{u_{all}d_{all}})$ among u_1, d_1, u_{all}, and d_{all} ((2), (3) and (4)).

$$r_{u_1d_1.u_{all}} = \frac{r_{u_1d_1} - r_{u_1u_{all}}r_{d_1u_{all}}}{\sqrt{1-{r_{u_1u_{all}}}^2}\sqrt{1-{r_{d_1u_{all}}}^2}} \tag{2}$$

$$r_{u_1d_{all}.u_{all}} = \frac{r_{u_1d_{all}} - r_{u_1u_{all}}r_{d_{all}u_{all}}}{\sqrt{1-{r_{u_1u_{all}}}^2}\sqrt{1-{r_{d_{all}u_{all}}}^2}} \tag{3}$$

$$r_{d_1d_{all}.u_{all}} = \frac{r_{d_1d_{all}} - r_{d_1u_{all}}r_{d_{all}u_{all}}}{\sqrt{1-{r_{d_1u_{all}}}^2}\sqrt{1-{r_{d_{all}u_{all}}}^2}} \tag{4}$$

Finally, the correlation coefficient between u_1 and d_1 eliminated the influences of u_{all} and d_{all} can be derived as follows: [2]

$$r_{u_1d_1.u_{all}.d_{all}} = \frac{r_{u_1d_1.u_{all}} - r_{u_1d_{all}.u_{all}}r_{d_1d_{all}.u_{all}}}{\sqrt{1-{r_{u_1d_{all}.u_{all}}}^2}\sqrt{1-{r_{d_1d_{all}.u_{all}}}^2}} \tag{5}$$

[2]Such a partial correlation coefficient is calculated for every pair of two request-types between two tiers in the n-tier system for the processes in the next section.

C. Eliminating the Influence of Coinvoked Request-Types

To eliminate the influence of coinvoked request-types, which is illustrated in Fig. 4, from the correlation between the request-frequency time-series data of a request-types U1 (u_1) and that of a request-types D1 (d_1), the first step is selecting the candidates of influencing the two request-types. Since the following processes require a high amount of calculation, the number of candidates need to be limited to small. We again adopt partial correlation analysis to eliminate the influence of coinvoked request-types, and Equation 1 shows the delta between a partial correlation coefficient and the original correlation coefficient become negligible when both of the correlation coefficients r_{xz} and r_{yz} are small (that is, the third variable have small correlation with the two target variables). Thus, the candidates of influencing request-type can be limited to those which request-frequency time-series have a certain amount of correlation with either u_1 or d_1 described as follows:

$$\begin{aligned} T_{candidate} &= \{t_i \mid t_i \in T_{all} \land \\ &\quad (has\_corr(t_i, U1) \lor has\_corr(t_i, D1)) \land \\ &\quad (tier(t_i) \text{ is upper than } tier(D1))\} \end{aligned} \tag{6}$$

where T_{all} is the set of all request-types in the n-tier system, $has\_corr(t_1, t_2)$ shows the request-frequency time-series data of the request-type t_1 have correlation with that of the request-type t_2 [3], and $tier(t)$ shows the tier of the request-type t.

The second step is calculating the partial correlation coefficient between u_1 and d_1 by removing the influence of each candidate request-type one by one. As the initial correlation coefficients ($R_{initial}$) among the target request-types U1, D1 and all the candidate request-types ($T_{initial}$), the correlation coefficients obtained in the previous section are used.

$$T_{initial} = \{t_i \mid t_i \in T_{candidate} \cup \{U1, D1\}\} \tag{7}$$

$$R_{initial} = \{r_{t_x t_y} \mid t_x, t_y \in T_{initial}\} \tag{8}$$

For a request-type $t_r \in T_{candidate}$, the adjusted correlation coefficients after eliminating the influence of the request-frequency time-series of the request-type t_r are shown as follows:

$$T_{adjusted} = \{t_i | t_i \in T_{initial} \land t_i \neq t_r\} \tag{9}$$

$$\begin{aligned} R_{adjusted} &= \{r_{t_x t_y . t_r} | t_x, t_y \in T_{adjusted}, \\ &\quad r_{t_x t_y . t_r} = \frac{r_{t_x t_y} - r_{t_x t_r} r_{t_y t_r}}{\sqrt{1 - {r_{t_x t_r}}^2}\sqrt{1 - {r_{t_y t_r}}^2}}, \\ &\quad r_{t_x t_y}, r_{t_x t_r}, r_{t_y t_r} \in R_{initial}\} \end{aligned} \tag{10}$$

After repeating this elimination ((9) and (10)) until $T_{candidate}$ becomes empty, the adjusted correlation coefficient between the target request-types U1 and D1, eliminated the influence of all request-types in $T_{candidate}$, can be obtained in $R_{adjusted}$.

D. Correlation Testing

The final step of the invoking relation estimation is the correlation testing on the adjusted correlation coefficient obtained through the previous two sections to judge whether the two target request-types U1 and D1 can be accepted as an invoking relation or not. [4] In the case the two request-types have correlation with a high significance level α=0.01, they are determined as having an invoking relation.

A value of correlation coefficient can be tested strictly using Student's t-distribution (Student's t-test) [10]. The following equation briefly explains the Student's t-test:

$$\begin{cases} \text{Accept as correlated} & : \ (r >= \frac{t}{\sqrt{dof + t^2}}) \\ \text{Reject (uncorrelated)} & : \ (otherwise) \end{cases} \tag{11}$$

where r is the correlation coefficient to test, dof is the degree of freedom (which equals the sample number $n - 2$), t is the test statistic for Student's t-test derived by a function which includes dof as a variable. It is noted here that the right-hand member of the above conditions becomes small when the number of samples increases. For example, with the sample number of 1000 and the significance level $\alpha = 0.01$ (this is the value used in our implementation), the minimum correlation coefficient to be accepted is only 0.081421.

The following provides the reasoning why we adopt such a strict testing for our invoking relation estimation. The main assumption of our method is that the request-frequency time-series of a request-type and that of another request-type should show high correlation if the two request-types are in an invoking relation. [5] In practice, however, the correlation coefficient between two request-types stays in small values closer to zero rather than one even the two request-types are in an invoking relation. There are mainly two reasons which explain such low positive correlation cases. First, a request-type can be invoked by multiple request-types in upstream tiers. For example, an SQL request-type in a database server tier can correspond to multiple HTTP request-types in an web/application server tier. Second, there are cases that a request-type is irregularly invoked by a request-type in an upstream tier. In both cases, the correlation coefficients become much lower than one and closes to values for no correlation cases. In order to correctly extract invoking relations from such low correlation coefficient cases, the strict correlation testing with such a high significance level is required as described in this subsection.

III. Response Delay Monitoring System

Using the invoking relations collected in the previous section, we developed a response delay monitoring system for an n-tier system, which can detect response delays of each request-type and can diagnose the invoking relations among detected delayed request-types to visualize propagation relations among the delays. Fig. 5 shows the overview of the developed monitoring system. The system consists of two phases, which are the monitoring phase for real-time monitoring of a target n-tier system and the learning phase to generate two types of models used in the monitoring phase.

The learning phase consists of two types of modeling. One is the response time modeling which provides the border

[3]For this condition, we test the partial correlation coefficient obtained in the previous section with a relatively low significance level α=0.10 since this testing just aims to reduce the amount of calculation.

[4]Although these consecutive subsections only explain the detailed steps to judge the invoking relation between an example pair of request-types U1 and D1, it is needless to say the same steps are necessary for each pair of request-types in the target n-tier system.

[5]Moreover, if a request-type in a downstream tier is invoked only by a single request-type in an upstream tier and the invoker always invokes the request-type, the correlation coefficient between their request-frequency time-series should close to one.

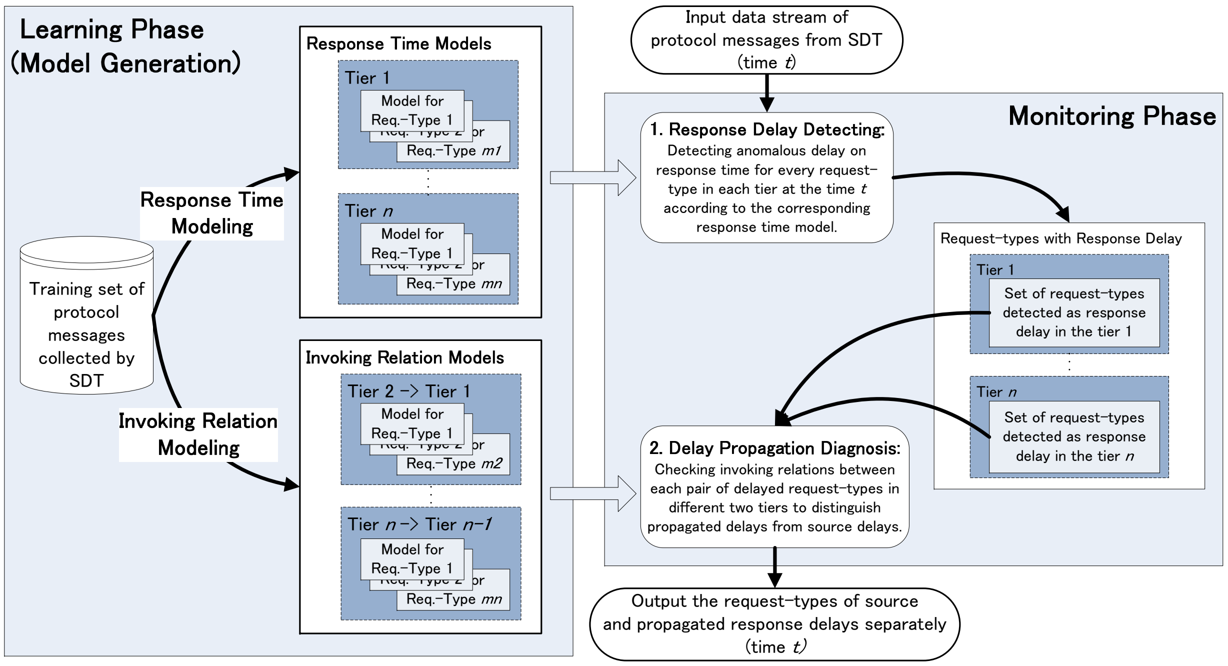

Learning Phase (Model Generation)
Training set of protocol messages collected by SDT
Response Time Modeling
Invoking Relation Modeling
Response Time Models
Tier 1
Model for Req.-Type 1
Req.-Type m1
Tier n
Req.-Type mn
Invoking Relation Models
Tier 2 -> Tier 1
Req.-Type m2
Tier n -> Tier n-1
Input data stream of protocol messages from SDT (time t)
Monitoring Phase
1. Response Delay Detecting: Detecting anomalous delay on response time for every request-type in each tier at the time t according to the corresponding response time model.
Request-types with Response Delay
Set of request-types detected as response delay in the tier 1
Set of request-types detected as response delay in the tier n
2. Delay Propagation Diagnosis: Checking invoking relations between each pair of delayed request-types in different two tiers to distinguish propagated delays from source delays.
Output the request-types of source and propagated response delays separately (time t)

Fig. 5: Overview of the response delay monitoring system

between *normal* and *delayed* for the response time of each request-type based on its usual statistical trend. The other is the invoking relation modeling which provides invoking relations among request-types. The invoking relations help judging whether a delayed long response time of a request-type in a tier is the result of delay propagation of its invoking request-type in a downstream tier.

The inputted data of both the two phases are provided by a passive trace monitoring tool *SDT* [1] as shown below.

Learning Phase: The phase requires a sequence of protocol messages collected by SDT during a certain period of time as a training data set.

Monitoring Phase: The phase requires a data stream of protocol messages supplied by SDT. Specifically, the protocol messages collected in a given length of time window (e.g., 1 minute) are inputted into this phase, and its output shows the response delay situation in the target n-tier system at that time window.

The passive trace monitoring tool SDT is introduced in Section III-A, while the details of the two modeling in the learning phase are explained in Section III-B and III-C respectively, and finally Section III-D explains the monitoring phase.

A. Data Collection: Passive Trace Monitoring Tool "SDT"

To collect accurate message interactions among multiple tiers in a target n-tier system, we use the passive trace monitoring tool SDT developed by FUJITSU FSAS [1]. The processing of SDT includes the following three steps as shown in Fig. 6: 1) Collecting all IP packets sent and received in

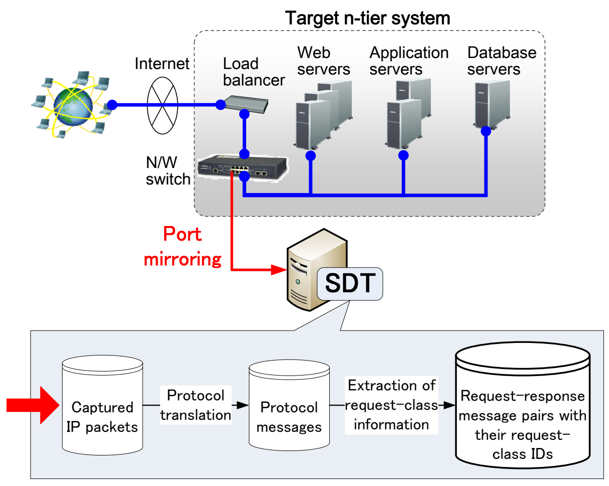


Fig. 6: Passive trace monitoring tool "SDT"

the target n-tier system by using port mirroring function of network switches; 2) Translating the captured IP packets to protocol messages (e.g., HTTP, MySQL, and so on) exchanged between component servers; and 3) Extracting identification information from each protocol message (e.g., URL for an HTTP request) so as to classify it into several tens (or hundreds, thousands) of request-types.

There are several merits to use such a passive trace monitoring tool to monitor an n-tier system instead of typical agent-based or logging-based monitoring tool deployed or installed into each server of the system. First, its monitoring overhead is negligible since the monitoring system is physically separated

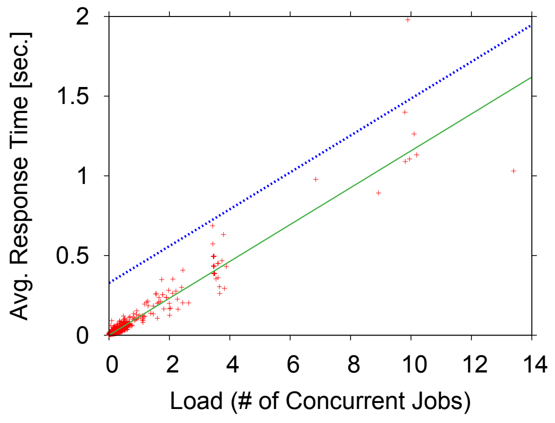


Fig. 7: Example to illustrate how the response time modeling derives the upper limit of response time for a request-type from the scattergram of load and response time for the request-type.

from the target system. Second, it requires no modification to the application (or server) source code.

B. Response Time Model

The response time model provides the border between *normal* and *delayed* in response time of each request-type, based on its usual statistical trend. One model is provided for every request-type in each tier of the target n-tier system.

The upper limit of adequate response time for a request-type changes depending on the instantaneous load of the server. In general, under the situation that no hardware resource is fully saturated, the response time of a request-type in a server increases near-linearly as the load in the server increases due to waiting time for the concurrent requests. Fig. 7 illustrates how the upper limit of response time for a request-type is decided during the response time modeling. In this figure, each point represents a pair of load and response time aggregated in a small time window (e.g., 1 second in our implementation), meanwhile the solid line shows the linear approximation of the points. The upper limit of response time is derived as the upper limit of the confidence interval of the points shown as the dotted line in the figure.

C. Invoking Relation Model

The invoking relation model provides invoking relations between request-types in the target n-tier system. Specifically, an invoking relation model is provided for every request-type in each tier except the upmost tier, providing all request-types in upstream tiers which invoke the target request-type. Since there is no explicit information to distinguish such invoking relations in transaction messages exchanged among tiers [6], we obtain the model by estimating such relations through correlation analysis as described in Section II.

[6]For example, there is no information to connect an HTTP message received on a web server with a subsequent SQL message sent from the web server to a database server even the two request-types are in the same transaction.

D. Response Delay Monitoring Phase

The monitoring phase takes a data stream of protocol messages supplied by SDT as its input while outputs all the request-types on which response delay is detected in each time window, using the two types of models pre-generated in the learning phase. The monitoring phase consists of the following two steps.

At the first step, response delay of each request-type is examined. For the examination, the average response time of each request-type and the average number of concurrent processes in the server are aggregated in the time window *t*. For every request-type in each tier, response delay of the request-type is examined by corresponding with the response time model for the request-type. Each response time model is a linear function of the load in the server and the response time of the request-type, providing the upper limit of response time which is determined as normal according to the instantaneous load value at that moment. For each time window *t*, the request-types detected as delayed are recorded for the next step.

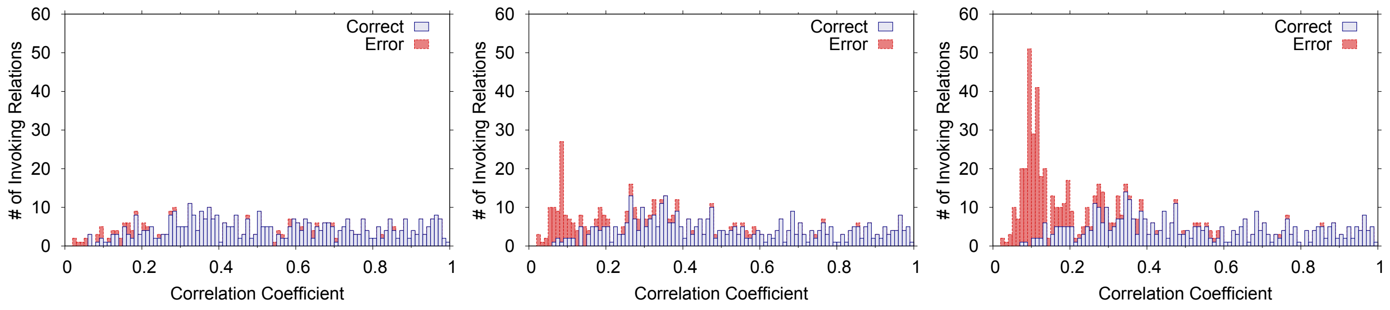
Then the second step determines in which request-types their response delays are affected by the response delays happened in downstream tiers through response delay propagation via invoking relations. In the diagnosis, the invoking relation model which corresponds to each delayed request-type is referenced to check whether there is any invoking relations between the request-type and another delayed request-type in an upstream tier. If two request-types have an invoking relation and both of which are detected as delayed at the same time window *t*, it is determined that the response delay of the request-type in the upper tier is caused by that of the other request-type in the lower tier. In that case, an alert for response delay is issued to the operations manager of the target n-tier system only for the request-type in the lower tier, reducing unnecessary alerts issued to the operations manager.

IV. Experimental Evaluation

In this section, we evaluate our method by using a real in-company two-tier system as the target system. First of all, the target system is introduced in Section IV-A. Then the accuracy of our invoking relation estimation method is evaluated by compared with manually extracted invoking relations in Section IV-B.

A. Target n-Tier System for the Evaluations

As the target n-tier system to evaluating our invoking relation estimation method, we selected an in-company two-tier system summarized in Table I. The system is an employee information system including an attendance management, a traveling expense management, and so on, for the whole employee of our company, which has over 100,000 users and averagely 2,300 concurrent users on weekdays. The system has anomalous two-tiers, the first tier consists of three load-balanced web/application servers while the second tier consists of one application server and two database servers. 3075 request-types are executed in the web/application servers with a workload of about 17 thousand requests per minute, while 1102 request-types are executed in the database servers with a workload of about 70 thousand requests per minute. We have manually extracted the invoking relations of the system with



(a) Our estimation method *(b) Our estimation method without eliminating coinvoked requests influence* *(c) Normal correlation analysis (as a typical existing method)*

Fig. 8: Distribution of the correlation coefficients in the invoking relations obtained by our estimation method with(8a)/without(8b) eliminating the influence of coinvoked requests, compared with a typical existing method (8c). This comparison clarify the significant effect of our two techniques (shown in Section II-B and II-C respectively) on improving estimation accuracy.

TABLE I: Target in-company two-tier system for the experimental evaluations

Tier	Server role	Protocol	# of node	# of request-types	Request rate (req/min)
1	Web/Application	HTTP	3	3075	17k
2	Application	IIOP	1	26	6k
2	Database	Symfo-WARE	2	1102	70k

TABLE II: Precision and recall of the estimated invoking relations comparing with manually extracted invoking relations

Type	Precision	Recall	Recall (frequency weighted)
Our estimation method	90.6%	49.8%	88.2%
Our estimation method w/o eliminating coinvoked requests influence	73.9%	47.7%	87.4%
Correlation analysis (as an existing method)	56.2%	47.8%	87.5%

the help of our previously developed other tools. This manually extracted invoking relations are used as baseline for the true invoking relations through the evaluation.

B. Accuracy of Invoking Relation Estimation

Table II presents the result of accuracy evaluations on the invoking relations obtained by our method, on that obtained by our method without eliminating the influence of coinvoked requests (described in Section II-C), and on that obtained by just correlation analysis and correlation testing without the two techniques described in Section II-B and II-C, by comparing with the manually extracted invoking relations as correct answers. The data used in these evaluations were collected from the target system during 7 hours (9:00am–4:00pm) on March 5th 2012, and aggregated into 25200 time windows of one second length each, that is, the request-frequency of each request-type is aggregated in every time window. Thus, there are maximally 25200 samples in the correlation analysis which enables a very small correlation coefficient value of less than 0.1 to be be accept as "correlated" in a correlation testing.

Here, we calculate the recall and the precision as follows:

$$recall = \frac{|R_{estimated} \cap R_{manual}|}{|R_{manual}|}, \quad (12)$$

$$precision = \frac{|R_{estimated} \cap R_{manual}|}{|R_{estimated}|}, \quad (13)$$

$R_{estimated}$: *the set of estimated invoking relations*,
R_{manual} : *the set of manually extracted invoking relations*.

The table has two types of recall, one is just as the above definition while the other is weighted by the request-frequency of each request-type, in order to reflect the condition that invoking relations for rarely executed request-types have less importance for the delay propagation analysis in our response delay monitoring system.

The result in the table clearly shows the effectiveness of our techniques for invoking relation estimation which decrease spurious correlation cases caused by the two factors addressed in this paper and increase the precision of estimation. The second and the third cases in Table II further show each of the two techniques in our estimation method correctly performs their roles on improving accuracy.

To provide further understanding how our estimation method provides the drastic gain on precision, the distributions of correlation coefficients in the three cases are compared in Fig. 8. This comparison clarifies that the correlation analysis without adjusting the two factors' influences (Fig. 8c) generates a lot of small (but large enough to accept as "correlated" in correlation testing) correlation coefficients around 0.05 to 0.2, which are scored as errors (that is, no invoking relations). Comparing the result with the second case (Fig. 8b) and that

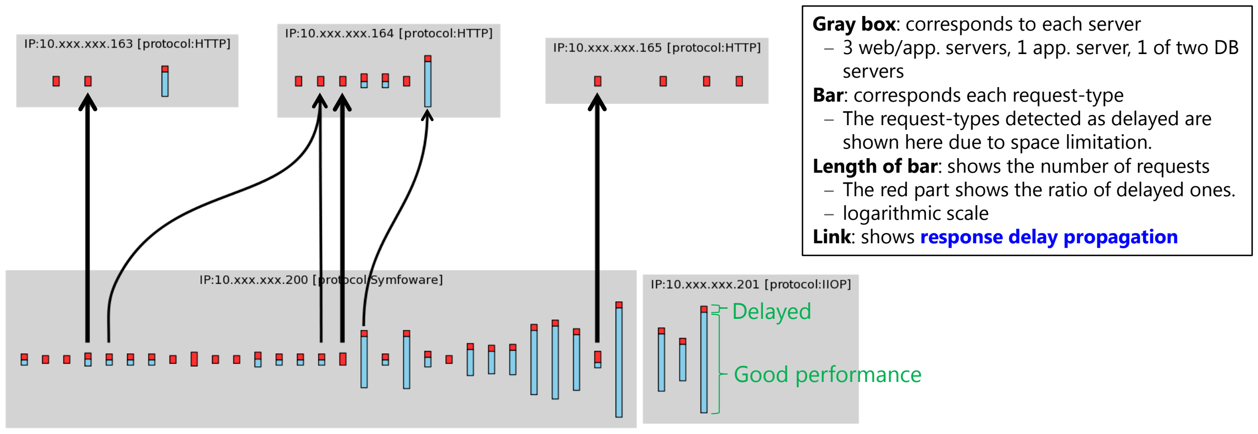

IP:10.xxx.xxx.163 [protocol:HTTP]
IP:10.xxx.xxx.164 [protocol:HTTP]
IP:10.xxx.xxx.165 [protocol:HTTP]
IP:10.xxx.xxx.200 [protocol:Symfoware]
IP:10.xxx.xxx.201 [protocol:IIOP]
Delayed
Good performance
Gray box: corresponds to each server
– 3 web/app. servers, 1 app. server, 1 of two DB servers
Bar: corresponds each request-type
– The request-types detected as delayed are shown here due to space limitation.
Length of bar: shows the number of requests
– The red part shows the ratio of delayed ones.
– logarithmic scale
Link: shows response delay propagation

Fig. 10: The result of response delay propagation analysis provided by our developed response delay monitoring system during one minute starting from 8:01am on April 4, 2014.

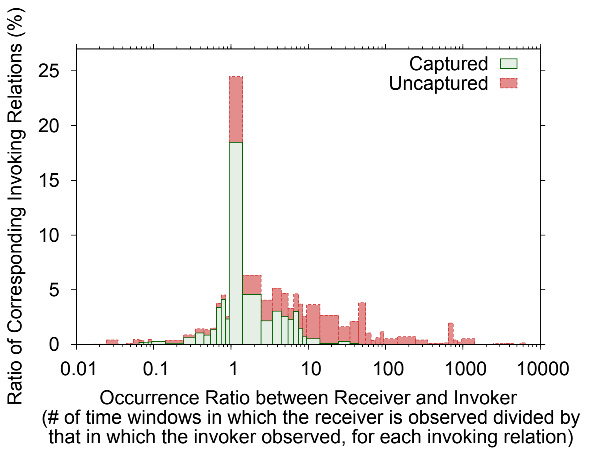


Fig. 9: Analysis of the manually extracted invoking relations by the observed time windows number of receiver and invoker request-types, explaining the reason of the modest recall value.

of our estimation method (Fig. 8a), it is derived that the small correlation coefficients are caused by spurious correlations caused by the two factors, which are eliminated by the two techniques in our estimation method respectively.

C. *Why is the Recall Modest?*

While our estimation method shows enough high recall value weighted for the purpose of our response delay monitoring, someone may wonder the reason of the modest recall value without weighted. In order to explain the reason, the ratio between the number of time windows the receiver request-type is observed and that the invoker request-type is observed for the manually extracted invoking relations is shown in Fig. 9. Each bar in the graph is divided into two areas, one denotes the ratio of invoking relations successfully captured by our estimation method while the other denotes those failed to be captured by our method. This result shows once the ratio exceeds 10, such invoking relations are rarely captured by our estimation method (that is, the correlation coefficients are too small to be accepted by correlation testing). Such a case happens when a popular request-type in a downstream tier is invoked by many request-types in an upstream tier and some of the request-types in the upstream tier are moderately invoked. However, as the weighted recall value in Table II shows, such request-types are minority according to their request-frequency and have only a limited impact on our response delay propagation analysis.

D. *Response Delay Propagation Analysis*

To illustrate the effectiveness of our response delay propagation monitoring system, we show an one-minute snippet of response delay propagation analysis data starting from 8:01am on April 4, 2014 (see Fig. 10). The monitoring system detected 46 request-types in 5 servers as delayed during the one minute (14 request-types and 32 request-types from the first tier and the second tier, respectively). While 14 request-types were delayed in the first tier, 5 of them were identified as the results of delay propagation caused by the 6 request-types in the second tier with which they have invoking relations. We note that the number of request-types executed during this one minute was much larger than that in the figure since the figure only shows the request-types detected as delayed for the purpose of human-readability.

V. Related Works

Techniques which extract causal relations from IP packet traces in black-box IT systems have been proposed in previous research for performance anomaly diagnosis [2], [3], [5]. Aguilera *et al.* [2] obtain causal relations among components in an n-tier application to find high-impact causal path patterns on the performance of the system. For the purpose, the technique introduced in the paper reconstructs every transaction trace by stochastically determining each parent-child pair of nested request-calls[7] from parallel execution of nested request-calls. However, this approach experiences

[7]It means each pair of an invoking request and the invoked request, in our words.

degradation of accuracy on determining nested request-calls when average response time increases and trace parallelism increases. On the other hand, our approach is robust to high parallelism of request execution by strictly applying statistical analysis on estimating invoking relations among request-types. Sherlock [3] and Orion [5] correlates a response delay in a networked component with that in another networked component. In other word, they aim to directly detect "delay propagations" among networked components instead of detecting them through invoking relations as we do. This approach is unsuitable to analyze response delay propagations among request-types in an n-tier system since a response delay in that case can indeterministically propagate to others through hardware/software resource dependences [13]. Our estimation method instead correlates request-frequencies of request-types which are deterministic through invoking relations.

Other studies also use packet trace monitoring for performance analysis of n-tier applications [12], [14]. Magpie [4] focuses on extraction of request-types and correlating them with resource consumption for performance prediction. Spectroscope [11] identifies anomalous requests by comparing request-flows between "problem" time windows and "non-problem" time windows. EtE [8] analyzes captured packets for end-to-end performance monitoring of networked services. Comparing with them, we focus on identifying response delay propagations among request-types inside of an n-tier application rather than detecting anomalous response degradation.

For performance analysis of an n-tier system, many studies correlate resource consumption of the system such as CPU consumption or disk I/O with the system's performance behavior [6], [7], [15]. We use correlation analysis in a decidedly different manner.

VI. Conclusions

In order to analyze the propagation relations among delayed request-types in an n-tier system, first we proposed a novel method for highly accurate estimation of invoking relations between a request-type executed in a tier and one executed in another tier in the system (Section II). The estimation method is based on correlation analysis between the request-frequency time-series of a request-type and that of another one in a different tier, and eliminates the spurious correlations caused by two factors (overall time trend and coinvoked request-types) to enable hair-splitting correlation testing precisely distinguish request-types which have an invoking relation between them (as evaluated in Section IV-B).

Then we showed the overall architecture of our developed response delay monitoring system which can diagnose response delay propagation among request-types and can distinguish root causes of response delay from propagated delays (Section III). We implemented the monitoring system and are using it for daily monitoring of the large in-company two-tier system in real. The snapshot of the response delay propagation analysis shown in Section IV-D was taken during the real daily monitoring.

One of our future works is to sophisticate the response time model by employing existing efforts on performance modeling, which is beyond the scope of this paper.

References

[1] Fujitsu FSAS, "Service tool SDT: System Diagnostic Tool". http://jp.fujitsu.com/group/fsas/services/operations/system/network-lcm/tools/#ns02 (Japanese only).

[2] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen. Performance debugging for distributed systems of black boxes. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 74–89, 2003.

[3] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies. In *Proceedings of the ACM SIGCOMM 2007 Conference*, pages 13–24, 2007.

[4] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using magpie for request extraction and workload modelling. In *Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation (OSDI'04)*, volume 4, pages 259–272, 2004.

[5] X. Chen, M. Zhang, Z. M. Mao, and P. Bahl. Automating network application dependency discovery: Experiences, limitations, and new solutions. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI'08)*, pages 117–130, 2008.

[6] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni. Automated anomaly detection and performance modeling of enterprise applications. *ACM Transactions on Computer Systems (TOCS)*, 27(3):6, 2009.

[7] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *Proceedings of the 6th USENIX Conference on Operating Systems Design and Implementation (OSDI'04)*, pages 231–244, 2004.

[8] Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat. EtE: Passive end-to-end internet service performance monitoring. In *USENIX Annual Technical Conference, General Track*, pages 115–130, 2002.

[9] L. Hu, K. Schwan, A. Gulati, J. Zhang, and C. Wang. Net-cohort: Detecting and managing vm ensembles in virtualized data centers. In *Proceedings of the 9th international conference on Autonomic computing (ICAC 2012)*, pages 3–12. ACM, 2012.

[10] N. A. Rahman, A. S. Macpherson, K. A. Rosemblatt, and N. P. Appelbaum. *A course in theoretical statistics: for sixth forms, technical colleges, colleges of education, universities*. Charles Griffin and Company, 1968.

[11] R. R. Sambasivan, A. X. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing performance changes by comparing request flows. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11)*, pages 43–56, 2011.

[12] B. H. Sigelman, L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. Dapper, a large-scale distributed systems tracing infrastructure. In *Google Technical Report*, 2010.

[13] Q. Wang, Y. Kanemasa, M. Kawaba, and C. Pu. When average is not average: large response time fluctuations in n-tier systems. In *Proceedings of the 9th international conference on Autonomic computing (ICAC 2012)*, pages 33–42. ACM, 2012.

[14] Q. Wang, Y. Kanemasa, J. Li, D. Jayasinghe, T. Shimizu, M. Matsubara, M. Kawaba, and C. Pu. Detecting transient bottlenecks in n-tier applications through fine-grained analysis. In *Proceedings of the 33rd IEEE International Conference on Distributed Computing Systems (ICDCS 2013)*, pages 31–40, 2013.

[15] Q. Zhang, L. Cherkasova, G. Mathews, W. Greene, and E. Smirni. R-capriccio: a capacity planning and anomaly detection tool for enterprise services with live workloads. In *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware (Middleware 2007)*, pages 244–265, 2007.