

Towards Noise-Tolerant Network Service Diagnosis

Yao Zhao
EECS Department
Northwestern University
{yzhao}@cs.northwestern.edu

1. INTRODUCTION

The modern Internet is a large-scale distributed system composed of many complex, inter-operating sub-networks. Today, even a sub-network such as an enterprise network or an ISP backbone network may have tens of thousands of network entities (*e.g.*, hosts, servers, routers, cables and so on). Any entity in Internet may be faulty, *e.g.*, out of order or of poor performance. The large amount of entities and the complicated (direct or indirect) dependencies bury the real faulty entities deeply in many cases. Naturally one may think that some butterfly effect also exists in the esoteric Internet system. Thus, when a network manager is asked, “Why is the network slow now?” the most frequent answer is, “I’m not sure.”

A straightforward troubleshooting approach for Internet is to install monitors on every entity in the distributed system. The monitors then generate alerts when they detect faults on the monitored entities. Although simple and powerful, this approach is not suitable in many cases, *e.g.*, when not all entities can be monitored or monitoring on each entity separately cannot tell the fault efficiently and accurately. In enterprise network service diagnosis, although monitors can be installed on every server, and they monitor the OS level and networking status, it is still hard to translate such metrics to the server performance. Therefore, to diagnosis such systems, people tend to use the measurements from the end clients and then correlate the experience of the end clients to diagnosis the internal system [1–3]. Usually the end clients measure the *request paths*, which embody the control flow, resources, and performance characteristics associated with servicing requests. Roughly speaking, there are three components in this troubleshooting problem: model, monitoring and diagnosis.

- *model*: The model defines the extrinsic observations and intrinsic faulty problems as well as the relationship between them. For example, in [1], the dependency graph models the relationship between the root cause nodes (*e.g.* routers, web servers, database servers and so on) and observation client nodes.
- *monitoring*: The monitoring component is to collect the observations for the next step of diagnosis. Generally, monitoring approaches can be active and passive. While passive monitoring does not introduce additional overhead to the network services, a challenging problem is how to understand the service performance from the sniffed network traffic or any other host information.
- *Diagnosis*: The diagnosis component identifies the root cause of the faulty observations. Obviously, the monitoring and diagnosis are tightly coupled. The design of monitoring should aim to help the diagnosis and the diagnosis algorithms should try to get optimal diagnosis result out of the monitoring data.

While the pilot work, Sherlock [1], shows a promising framework for diagnosing multiple network applications simultaneously in large scale enterprise networks, we observe many research problems left for deeper studies. In our current research stage, we focus on the diagnosis problem under such framework, and leave other research topics as our future work. The current diagnosis algorithm, Ferret, in [1] has a critical assumption on the number of simultaneous faulty roots in the network, which is not generally true in large and complex networks. We propose a Gibbs sampling based inference algorithm

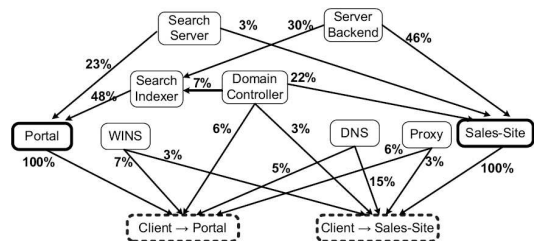


Figure 1: An example of the dependency graph.

that scales to very large networks with unlimited simultaneous faulty roots. Furthermore, since noise is nearly inevitable in the real systems, we focus on the design of noise-tolerance mechanism to apply in the Gibbs sampling. Specifically, we propose two noise-tolerance approaches for different noise types in observations.

2. BACKGROUND

2.1 The Dependency Graph

Sherlock [1] proposed a dependency graph model for network service diagnosis. Figure 1 shows an instance of the model involving one client and two services. For example, when the client is going to visit the web portal, the client machine may query the DNS for the web portal IP, or may not if previous DNS query is cached. The probability on the corresponding edge shows the probability that the client action will depend on the DNS (*e.g.* 5% in Figure 1).

2.2 Ferret

In Sherlock, the diagnosis algorithm applies the Occam’s razor principle and only considers a small number of simultaneous faulty components of the services. Ferret [1] assumes the simultaneous faulty components in the enterprise network are rare, *e.g.*, no more than 2 in most cases. Then it enumerates all the possible combinations of the faulty components and uses maximum likelihood estimation to find out the likely faulty components. Therefore, Ferret’s computation complexity increases exponentially as the number of simultaneous faulty components considered by Ferret increases.

3. NOISE-TOLERANT INFERENCE USING GIBBS SAMPLING

3.1 Bayesian Inference using Gibbs Sampling

Gibbs sampling [4] is a special and widely used implementation of the Markov chain Monte Carlo method (MCMC). Take our network service diagnosis as an example. The basic idea is that at each iteration, the algorithm only consider changing the state of a single root cause node, while keeping all other root cause node states. The new state of the particular root cause node in the iteration is randomly chosen from all possible states with probabilities proportional to the calculated likelihood probabilities. It can be shown that the sequence

of samples comprises a Markov chain, and the stationary distribution of that Markov chain is just the joint distribution [4].

Let θ_i ($1 \leq i \leq d$) be the state of the i th root cause node. Suppose the random variable A_j ($1 \leq j \leq m$) is the state of the j th observation node, and A_j can be 0 (good), 1 (troublesome) or 2 (down). Given the dependency graph, it is easy to calculate the conditional probability $P_r(A_1, \dots, A_m | \theta_1, \dots, \theta_d)$ (also called likelihood function). In one iteration, we pick every root cause node one by one following the order of their index. When processing the i th root cause node, we fix the states of all the other root cause nodes and compute the conditional distribution of the i th root cause node, *i.e.*, $g(\theta_1, \dots, \theta_{i-1}, \cdot, \theta_{i+1}, \dots, \theta_d)$. For simplicity, let

$$P_k = g(\theta_1, \dots, \theta_{i-1}, \theta_i = k, \theta_{i+1}, \dots, \theta_d), k = 1, 2, 3.$$

Then we randomly select a new state for the i th root cause node, and the probability of being state k is proportional to P_k .

3.2 Noise-Tolerant Diagnosis

There are three possible noises in the system: 1) noise in the dependent graph; 2) noise in the observation; 3) noise in the prior distribution. In this paper, we only consider the noise in the observations. There are two kinds of noises in observation. First, the active probing client may have some problem and reports wrong measurement results. This can be caused by the software bug or configuration mistakes. Second, when a service is overloaded and becomes troublesome, the service can still serve some requests well and gives fast response. Therefore, two requests to a same overloaded server may get different response latency.

3.2.1 Observation Noises

In this section, we consider that some observation nodes are problematic and frequently report wrong monitoring results. Note, we expect such problematic observation nodes to be very rare, otherwise, the noise instead of the true observations will dominate the inference results.

Intuitively, we think the problematic observation node usually give observation results in conflict with normal observations. Therefore, the inferred root cause node states may not explain the observations very well or even with some inconsistency. On the other hand, if the bad observations are removed, the inferred root cause node states should explain the observations better. To describe how well the inferred root cause node states explain the observations, we can use the likelihood function $P_r(A_1, \dots, A_m | \theta_1^*, \dots, \theta_d^*)$ while θ_i^* is the inferred state of root cause node i . Therefore, for each observation node j , we compute two likelihood function with and without the observation node j . If the likelihood function increases significant after the removal of observation node j , the j th observation node is marked as ‘‘suspicious’’. Once an observation node is marked to be suspicious in many inference instances, the node is thought to be problematic.

3.2.2 Hidden Overloading Degree

Consider that the overloaded service may have different latency to different requests, we can enhance the dependency graph and add a hidden overloading degree for each root cause nodes.

For each root cause nodes N_i , let random variable o_i ($0 \leq o_i \leq 1$) be the overloaded degree for N_i , which means a request will be delayed for a quite long time with probability o_i .

o_i can be a continuous variable or a discrete variable. And the probability distribution function (or probability density function) is unknown to us. For simplicity, we consider o_i as a discrete variable between 0 and 1. In simulation, o_i can be $k/10$ ($k = 0, 1, \dots, 10$). When $o_i = 0$, it actually means the server is good and will never delay any request. Therefore, when assigning the prior probability distribution of o_i , $P_r(o_i = 0)$ should be much larger than $P_r(o_i > 0)$. Introducing the hidden overloading degree to Gibbs sampling

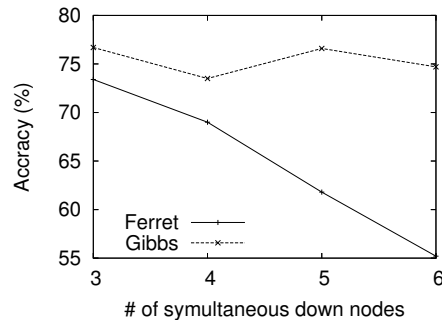


Figure 2: Accuracy of Ferret and Gibbs sampling.

does not change the algorithm much. It only increases the possible states of the root cause nodes and hence is easy to be adopted.

4. EVALUATION

We conducted some preliminary evaluations on our Gibbs sampling based inference and got promising simulation results at this stage.

We use the dependency graph inferred from the testbed used in [1]. There are in total 345 observation nodes and 355 root cause nodes in the testbed. In our simulations, we randomly select x simultaneous faulty (troublesome or down) root cause nodes, and then randomly generate the states of the observation nodes based on the computed likelihood function. In the preliminary simulations, we do not introduce any noise yet. Ferret considers all combinations of at most 3 simultaneous faulty roots in the simulation, because Ferret will run for hours considering 4 simultaneous faulty roots. To evaluate the inference accuracy, we do not use the standard false negative and false positive. Instead, we consider the $2x$ most likely faulty root cause nodes inferred by our Gibbs sampling based algorithm and Ferret, and the *accuracy* hence is the percentage of real faulty root cause nodes that are within the set of $2x$ inferred root cause nodes. In this way, we actually fixed the false positive of both algorithms and compare their false negative. We run the simulation 100 times. Figure 2 shows the accuracy of Ferret and our Gibbs sampling based algorithm as a function of different simultaneous down root cause nodes in the network. Clearly the accuracy of Ferret drops as the number of simultaneous roots increases, as Ferret cannot well deal with more simultaneous faulty roots than expected. Meanwhile, our Gibbs sampling based algorithm is not affected by the number of simultaneous faulty roots.

Our further analysis on the reason of missed faulty root cause nodes shows another big problem in the inference, *i.e.*, ambiguity problem. Actually, in the dependency graph, some root cause nodes have nearly the same effect on the observations. For example, imagine the two root cause nodes, the web server and the server’s first link to the network. It is hard to differentiate which of the two is down in many cases. We will further study on how to deal with the ambiguity problem in future research.

5. REFERENCES

- [1] 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 *ACM SIGCOMM*, 2007.
- [2] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, ‘‘Performance debugging for distributed systems of black boxes,’’ in *SOSP*, Oct. 2003.
- [3] P. Reynolds, J. L. Wiener, J. C. Mogul, M. K. Aguilera, and A. Vahdat, ‘‘Wap5: Black-box performance debugging for wide-area systems,’’ in *WWW*, May 2006.
- [4] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, *Markov Chain Monte Carlo in Practice*, Chapman and Hall, London, U.K., 1996.