Design of a Client-side Monitoring System

Learn to design a system to monitor the errors that don't reach our service.

We'll cover the following

A service has no visibility of the errors that don't occur at its infrastructure. Still, such failures are equally frustrating for the customers, and they might have to ask their friends, "Is the service X down for you as well?" or head to sites like Downdetector to see if anyone else is reporting the same issues. They might report the problem via a Tweet or some other communication channel. However, all such cases have a slow feedback loop. As a service provider, we want to detect such problems as quickly as possible to take remedial measures. Let's design such a system.

Initial design

To ensure that the client's requests reach the server, we'll act as clients and perform reachability and health checks. We'll need various vantage points across the globe. We can run a service, let's call it prober, that periodically

sends requests to the service to check availability. This way, we can monitor reachability to our service from many different places.

Prober checking availability of website

Issues with probers

We can have the following issues with probers:

- **Incomplete coverage**: We might not have good coverage across all autonomous systems. There are 100,000 unique autonomous systems on the Internet as of March 2021. It's not cost-effective or even possible to put those many probes across the globe. Country or ISP-specific regulations and the need for periodic maintenance are additional hurdles to implementing such a scheme.
- **Lack of user imitation**: Such probes might not represent a typical user behavior to explain how a typical user will use the service.

> **Note:** The initial design is based on active probing.

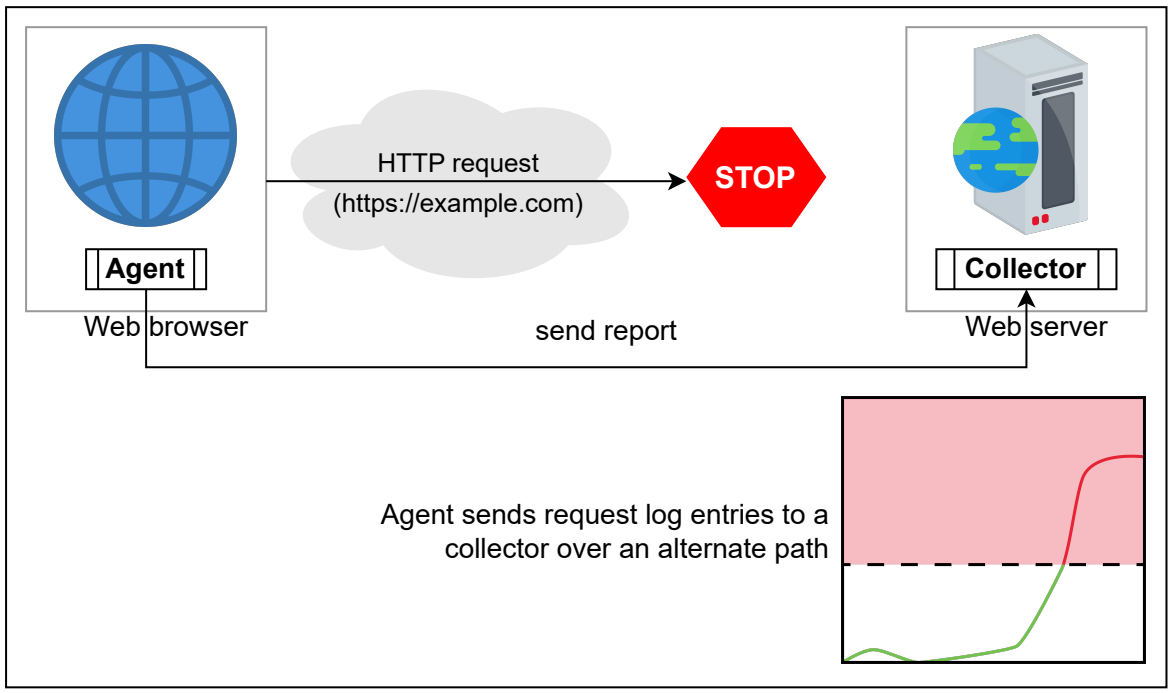
Improve the design

Instead of using a prober on vantage points, we can embed the probers into the actual application instead. We'll have the following two components:

- **Agent**: This is a prober embedded in the client application that sends the appropriate service reports about any failures.

- **Collector**: This is a report collector independent of the primary service. It's made independent to avoid the situations where client agents want to report an error to the failed service. We summarize errors reports from collectors and look for spikes in the errors graph to see client-side issues.

The following illustration shows how an agent reaches an independent collector when primary service isn't reachable:



Evaluating the agent's report to see a spike in errors

These collectors are a hierarchy of big data processing systems. We can place them near the client network, and over time, we can accumulate these statistics from all such localized sites. We'll use online stream processing systems to make such a system near real-time. If we're mainly looking for summary statistics, our system can tolerate the loss of some error reports. Some reports will be relative to the overall user population. We might say 1% of service users are "some." If we don't want to lose any reports, we'll need to design a system with more care, which will be more expensive.

Now, we'll solve the following concerns:

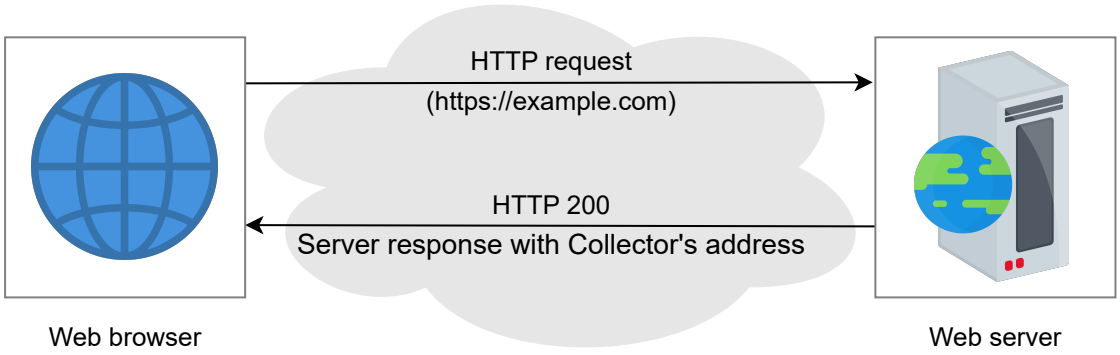
- Can a user activate and deactivate client-side reports?
- How do client-side agents reach collectors under faulty conditions?
- How will we protect user privacy?

Activate and deactivate reports

We'll use a custom HTML header to send appropriate information to the collectors. Though a client accesses the service via a browser, a specific browser should know about this feature to appropriately fill in the header information in the HTTP requests. For organizations that make browsers and provide services (for example, Chromium-based browsers), such features can be incorporated and standardized over time.

Another solution can be to use a client-side application that the service controls, and then we can easily include such headers over HTTP.

The client can fill in the request header if the client has already consented to that. The service can then reply with appropriate values for the policy and collection endpoints.



Activate and deactivate reports

Reach collectors under faulty conditions

The collectors need to be in a different failure domain from the web service endpoint that we're trying to monitor. The client side can try various collectors in different failure domains until one works. We can see a similar

pattern in the following examples. At times, we refer to such a phenomenon as being outside the blast radius of a fault.

If we want to see the reachability of an IP, we host the service on a different IP. If we monitor the availability of a domain, we host the collector on a different domain. And if we want to detect that an autonomous system route isn't hijacked, we host the service in a different autonomous system. Though, for last-mile errors, there isn't much we could do as a service provider. We might accumulate such events at the client side and report them on the next connectivity. A service can influence the remaining component failures.

Reaching Collectors Under Faulty Conditions

1.2.3.4 unreachable	Different server IP
Can't resolve example.com	Different domain
AS 1234 hijacked	Different ASN
CDN available	Different/no CDN
Last-mile problems	No readily available fall-back for the service

Protect user privacy

The human user who uses the client-side software should be in full control to precisely know what data is collected and sent with each request. The user should also be able to reactivate the feature any time they wish. If we use our client-side application (and not a browser application), we have a l of flexibility in what diagnostic could be included in the report. For a browser-based client, we can avoid the following information:

- We can avoid including traceroute hops to see a client to the service path. Users can be susceptible to their geographic location. It might be akin to collecting location information.
- We can avoid including which DNS resolver is being used. Again, details of DNS can leak some information about the location.
- We can avoid including round-trip-time (RTT) and packet loss information.

> **Note:** As a guiding rule, we should try to collect as little information as possible, and it must only be used for the specific purpose a user gave consent for.

Ideally, for a web-based client, we should only collect the information that's logged in the weblog when any request has been successful.

We shouldn't use any active probing except to test the service's standard functionality and report such probes' results. So, traceroute and RTT or packet loss information is excluded.

Any intermediary (like ISPs or middleboxes) can't change, add, or remove the error reporting mechanism due to encryption. Similarly, designated collectors are the only place where such data can go.

Conclusion

- In a distributed system, it's difficult to detect and respond to errors on the client side. So, it's necessary to monitor such events to provide a good user experience.
- We can handle errors using an independent agent that sends service reports about any failures to a collector. Such collectors should be independent of the primary service in terms of infrastructure and deployment.

Back

Focus on Client-side Errors in a Monitoring S...

Next

System Design: The Distributed Cache

Mark as Completed