November 2022

Backend Microservices Aware Server Overload Protection

n/a

Backend Microservices Aware Server Overload Protection

ABSTRACT

This disclosure describes techniques to provide server overload protection in microservices based computing systems. Per techniques of this disclosure, new incoming traffic is throttled at a server if it is determined that backend processes downstream of the server are overloaded. The techniques can mitigate performance degradation of server stacks while enabling optimal computing resource utilization. Metrics associated with overloading of the backend processes are monitored by the server. If it is determined based on the metrics that one or more backend processes are overloaded, the server performs throttling whereby part of the new incoming requests to the server are rejected and thereby rate limited. A cost function threshold, e.g., a running window average of a cost function such as queries per second received and successfully processed by the server, is determined. When backend processes are overloaded, new requests are accepted at the server only when an incoming cost function meets the cost function threshold.

KEYWORDS

- Server overload
- Request throttling
- Microservices
- Queries per second (QPS)
- Server latency
- Request retries
- Remote Procedure Call (RPC)

BACKGROUND

Many cloud-based software applications utilize a microservices architecture whereby a software application is designed as a set of services that run their own processes, and that communicate with one another, e.g., via application programming interfaces (APIs). Since each

service operates independent of other services, various services can become overloaded during operation based on the application workload and traffic patterns. Overloaded services can lead to particular computing devices such as servers becoming overloaded which can in turn lead to increased latency and/or error rate as well as disproportionally high usage of computing resources such as central processing units (CPU) and/or memory. In some cases, server overload can lead to cascading failures of one or more services as well as of a supported software application.

DESCRIPTION

This disclosure describes techniques to provide server overload protection in microservice based computing systems. Per techniques of this disclosure, new incoming traffic is throttled at a server if it is determined that backend processes downstream of the server are overloaded. The techniques can mitigate performance degradation of server stacks while enabling optimal computing resource utilization.

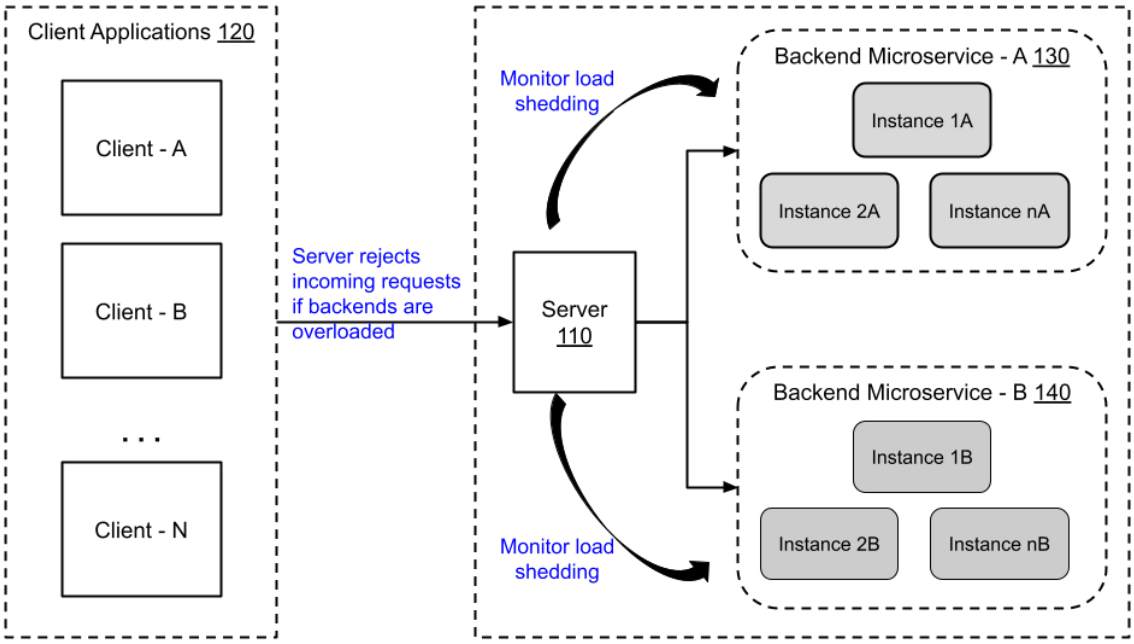

Fig. 1: Microservices based architecture

Fig. 1 illustrates an example of a microservices based computing system. As depicted in Fig. 1, a server (110) receives and processes incoming requests (traffic) from multiple client applications (120). The requests can be received asynchronously and can be associated with different levels of computing resource utilization. In this illustrative example, the server executes backend processes - backend microservice A (130) and backend microservice B (140) - to perform portions of a workload. The backend microservices (processes) can invoke multiple instances of a corresponding function to perform their portion of the workload.

Per techniques of the disclosure, metrics associated with overloading such as load shedding, re-tries, traffic rejection, etc. of the backend processes are monitored by the server. For example, a server can monitor whether the outgoing backend calls (e.g., remote procedure calls) are throttled and/or re-tried at a backend process. If it is determined based on the metrics that one or more backend processes are overloaded, the server performs a throttling operation whereby part of the new incoming requests to the server are dropped (rejected) and thereby rate limited.

Throttling can provide an additional advantage that results in that portions of the workload successfully completed by a microservice that is not overloaded can be utilized when another portion of the workload assigned to another overloaded microservice is subsequently completed, e.g., after a retry.

Rejection of new incoming workloads provides a signal to upstream processes, e.g., clients of the server to throttle their outgoing requests. Techniques of this disclosure can be implemented in a stackable manner since each server (task) can monitor the state of its outgoing (downstream) requests and implement throttling for upstream processes without a global central point for coordination. The techniques can also be utilized with any pre-existing throttling

mechanisms in a server that can be utilized for portions of the workload that are performed by the server (not assigned to backend processes).

Upon determination that backend processes are no longer overloaded, normal server operation can resume and work items that were not processed due to the throttling can be allocated.

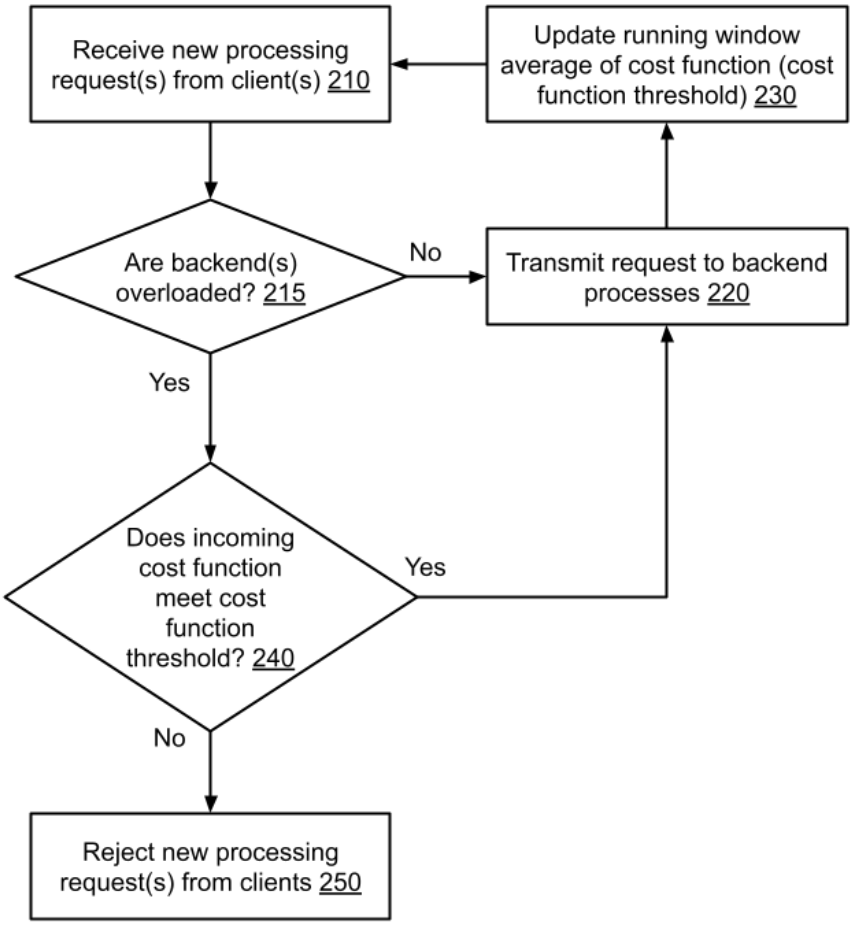

Fig. 2: Overload protection via throttling of new requests

Fig. 2 depicts an example method to provide overload protection via throttling of new requests, per techniques of this disclosure. New processing requests(s) are received (210) at a server from client application(s). It is determined (215) whether any of the backend processes of

the server are overloaded. For example, it may be determined whether any of the backend processes are rejecting traffic, throttling, performing re-tries, etc.

If it is determined that none of the backend processes are overloaded, the new processing request(s) are processed, and backend processes are called (220) to perform their portions of a workload. A cost function threshold, which is a running window average of a cost function, e.g., queries per second (QPS) received and success processed by the server, is determined (230). In this example, the cost function threshold is a running average value of a suitable cost function during normal operation of the server, e.g., when no backend processes are overloaded. The server continues to receive and service new processing requests.

If it is determined that a backend process is overloaded, an incoming cost function is determined (240) based on the new requests and compared to the cost function threshold. If the incoming cost function meets the cost function threshold (e.g., incoming cost function is less than the cost function threshold), no throttling is performed. The incoming requests are accepted and transmitted to the backend processes (220). However, if the incoming cost function does not meet the cost function threshold (e.g., incoming cost function is greater than or equal to the cost function threshold), the server performs throttling, and the new processing requests from the clients are rejected (250).

Optionally, various additional heuristics may be utilized for additional stability:

- The size of the averaging window for the cost function may be adjusted to achieve an optimal operation point for the reaction time and stability tradeoff.
- The cost function threshold may be adjusted to allow additional traffic (under backend overloaded conditions) to enable increased utilization of the backends, while allowing for a specified (tolerable) failure rate.

- The cost function threshold may be adjusted to account for backend work that was completed, but with retries.

CONCLUSION

This disclosure describes techniques to provide server overload protection in microservices based computing systems. Per techniques of this disclosure, new incoming traffic is throttled at a server if it is determined that backend processes downstream of the server are overloaded. The techniques can mitigate performance degradation of server stacks while enabling optimal computing resource utilization. Metrics associated with overloading of the backend processes are monitored by the server. If it is determined based on the metrics that one or more backend processes are overloaded, the server performs throttling whereby part of the new incoming requests to the server are rejected and thereby rate limited. A cost function threshold, e.g., a running window average of a cost function such as queries per second received and successfully processed by the server, is determined. When backend processes are overloaded, new requests are accepted at the server only when an incoming cost function meets the cost function threshold.