September 13, 2018

Client Request Scheduling to Reduce Server Load Spikes

Jay Gengelbach

Kirby Bohling

Client request scheduling to reduce server load spikes

ABSTRACT

Popular mobile apps have hundreds of millions of users. Consequently, servers that support such apps can receive as many as hundreds of thousands of user requests every second. Certain applications submit requests from a large number of devices such that the requests arrive at the server at nearly the same time. This causes a sharp spike in the number of user requests to be processed by the server and congestion throughout the network stack that can result in errors and dropped user requests. This disclosure presents techniques that schedule incoming user requests such that the histogram of load-versus-time is relatively flat and smooth. A flatter load-versus-time curve thus obtained results in higher system availability, lower error-rates, less strain on load-balancers, and enables a greater level of user satisfaction.

KEYWORDS

- Scheduling
- Load spike
- Randomized scheduler
- Weighted scheduler
- Load curve
- Mobile apps
- Simultaneous requests
- Load balancing
- High availability systems

BACKGROUND

Cloud-based service providers and servers that support popular software applications, e.g., mobile apps, receive requests from hundreds of millions of user devices. The requests generate hundreds of thousands of queries per second. Servers generally instruct clients to issue time-randomized requests, such that the load due to client requests received by a server follows a relatively smooth curve. However, large spikes in load are still observed at particular points in time.

For example, in applications that are triggered by time, e.g., calendar apps, smart assistant apps, etc., the very first second of an hour includes spikes that are many times larger than the average load. There are other reasons why load spikes arrive at particular times of the hour, e.g., alarms are often set at the top of an hour. Consequently, users resume using mobile devices close to the top of the hour. Other reasons for time-dependent load spikes include battery-saving or data-saving applications that prohibit data communication during most of the hour (or other time period) and perform data access within a narrow window of time.

A sudden spike in user requests creates a ripple of disturbance throughout the network stack. It creates problems for internet service providers or cellular networks. Such a spike can overload, even overwhelm, load-balancing infrastructure of the application server. It causes an increase in error rates for users of the application and a reduction in system availability, resulting ultimately in user dissatisfaction. However, a data center provisioned to serve spike-level loads that are much higher than loads at other times has high costs, and is not practical. Infrastructure providers benefit from smooth transitions between high and low levels of load (taking minutes to fully shift), and a flatter load-versus-time curve.

DESCRIPTION

Application servers typically perform randomization of scheduled background requests, e.g., by requesting client applications to make requests at times that are evenly smeared across an hour or day such that there isn't a single point in time with a large spike in requests. Some client applications are, for various reasons, invariably not able to send requests during scheduled slots. For example, an application on a mobile device under data-saver or battery-saver mode may be constrained to send updates at particular times, e.g., at the top of the hour. At the server, such requests result in an uneven time distribution of user requests.

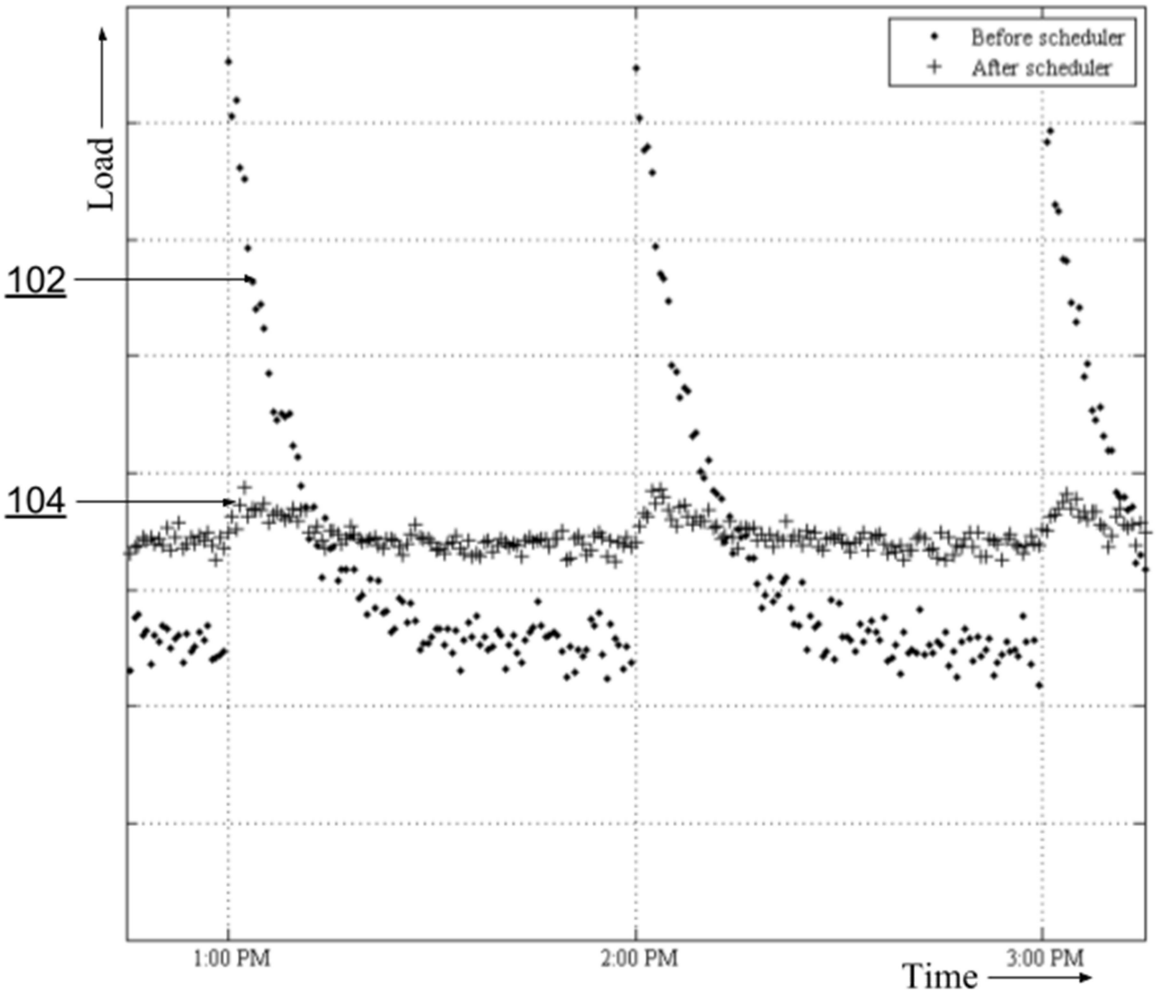

Fig. 1: Spikes in load-versus-time curve and reduction using scheduler

Fig. 1 illustrates load versus time at a data center, e.g., based on user requests received per second. The dotted curve (102) is a raw load-versus-time curve. Spikes in user requests are seen at the top of each hour, e.g., in the very first second of each hour. Such spikes occur despite server requests to clients to randomize their request times. The curve with crosses (104) is an illustrative load-versus-time curve after scheduling techniques, as described herein, are implemented. As seen in Fig. 1, the load-versus-time curve after the application of a scheduler is smoother, with gradual increases or decreases, and flatter, having a smaller peak-to-average ratio.

Per techniques of this disclosure, the server schedules user requests to shift traffic away from times of high load. Rather than using a uniform random generator to randomize user requests, the techniques use a weighted random number generator that avoids scheduling randomized traffic at times of high load.

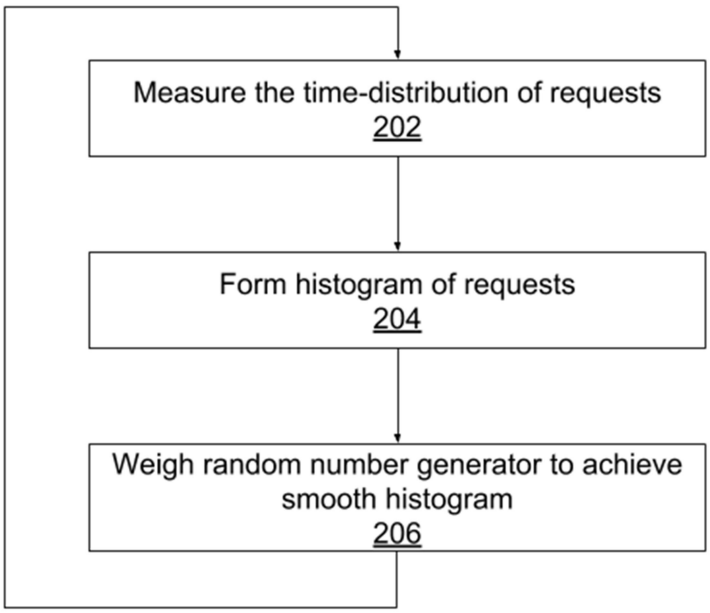

Fig. 2: Weighing a random number generator to reduce spikes in load

Fig. 2 illustrates the steps involved in reducing spikes in the load-versus-time curve. The distribution of user requests is measured across time (202), e.g. within each hour. Such a measurement forms a baseline that is used to form a histogram of user requests (204). The random number generator is weighted based on the histogram (206) such that the more active (spiky) parts of the histogram are avoided as scheduling slots in favor of the less active (flatter) parts of the histogram. Slots are assigned to clients based on the output of the random number generator. The operations are repeated to continually smoothen the load-versus-time curve.

The scheduling techniques described herein work even if only a fraction of clients actually adhere to the assigned slots. In effect, the spikes caused by clients that transmit requests outside assigned slots are absorbed by shifting the transmittals of clients that are able to adhere to assigned slots. Specifically, the techniques herein are independent of any ability of the server to control clients. Those clients that are unable to adhere to assigned slots are not brought under server control. It is only necessary that such clients be statistically predictable, and a small-enough fraction of the overall population. When the ratio of clients that adhere to the assigned slot to clients that do not is high, spikes are largely eliminated. In this manner, the back-end servers that serve requests from client applications continue providing service to both classes of clients.

The techniques are applicable to services with a relatively large amount of background server-client communications that can be scheduled at preferred times rather than in real time. The scheduling techniques described herein can be executed by client devices and/or servers. For example, once a server-side measurement of the request histogram is made, that histogram can be transmitted to client devices to configure the client devices to execute the weighted random number generator. In such an implementation, scheduling of slots is done by client

devices in a distributed manner. At a large-enough scale, e.g., user base of an application, the setting of scheduling slots need not be driven by a random number generator. Rather scheduling slots (tickets) may be handed out in a non-random manner. Further, instead of the scheduler targeting a flat or neutral load-versus-time curve, the scheduler can be implemented such that it attempts to generate spikes or swells at desired times during a day. In this manner, the techniques described herein use clients that are able to adhere to assigned slots to shape a load curve as necessary.

CONCLUSION

Due to a variety of reasons, certain times of the hour or day are observed to be associated a large number of user requests, e.g., from client applications sending requests to servers, that causes load spikes at servers. This disclosure provides techniques to reduce spikes in the load-versus-time curve at a server. The histogram of user requests versus time is determined at the server, and a random number generator is weighted to assign slots to client devices to make requests at times distinct from those associated with spikes. This results in load-versus-time curve that is flat or has a targeted shape. The techniques use clients that are able to adhere to assigned slots in order to shape a load curve with desired properties. A flatter load-versus-time curve thus obtained results in higher system availability, lower error-rates, less strain on load-balancers, and enables a greater level of user satisfaction.