Essay on *An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems* by Gan et al.

Aeilko Lübsen (57469)

May 3, 2022

In the paper *An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems*, the authors evaluate the implications microservices have on the performance of the different elements in the cloud computing stack. To do this, they developed a benchmark suite with different cloud computing applications, in which every request is timed to see where and how Quality-of-Service (QoS) violations originate.

Microservices offer a number of improvements over monolithic architectures, which is why they are increasingly popular in cloud computing settings, used by some of the biggest companies like Amazon, Netflix or Twitter. Their main advantage is their easy decomposability, simplifying development by only making each microservice responsible for a small part of the application's functionality. This also reduces the effort required to deploy, update and scale parts of the application independently. Additionally, microservices allow for heterogeneity in programming languages and frameworks only requiring a common API for the microservices to communicate with each other. Microservice architectures also simplify correctness and performance debugging. The goal of this paper is to evaluate the trade-offs that have to be made for these clear improvements. For this purpose, the authors developed six different end-to-end services of different cloud and edge services: a social network, an e-commerce site, a media service, a secure banking system and an IoT service for drone swarms. Every request in these systems is traced and centrally logged to determine the time it spends in each microservice. The implications of the architecture are measured on four different levels. First, the computer architecture itself is examined to see whether big or small cores are preferable, the performance of instruction caches and potential ways for hardware acceleration. Second, the networking and operating system implications are quantified, analyzing the time spent in the kernel and the fraction of time required for network processing. Third, the authors examine the influence of microservice architectures on cluster management to see how well they scale to prevent QoS violations. Fourth, the authors identify influences of microservice architectures on implementation questions by identifying bottlenecks in their service, quantifying the performance trade-offs between RPCs and RESTful APIs and exploring serverless programming frameworks for implementing their solutions. Finally, the authors use real application deployments to determine effects of scale that only arise with hundreds of users. To have an accurate comparison, the authors also implemented the social network and e-commerce applications in monolithic architectures.

On the hardware architecture question, the authors first determined that the cycles breakdown of the monolithic and microservice architecture are similar, however the former experience a slightly higher percentage of committed instructions due to a lower number of front-end stalls. The IPC of both architectures is also similar and consistent with previous studies of cloud applications. In contrast, microservices experience a significantly lower amount of i-cache misses than the monoliths, likely caused by the simplicity of the microservices themselves. Lastly, the influence of processor core frequency is examined, by both throttling the processor cores and also comparing to running the services on a different low-frequency processor. Counterintuitively, the microservices are much more sensitive to poor single-core performance. The authors hypothesize that this is caused by the much stricter time constraints for the single microservices themselves.

In the OS and networking area, the authors first analyzed the the number of cycles and instructions spent in kernel-mode, user-mode and libraries for each end-to-end service. Although the percentages of the time spent in each mode varies between the different services, all of them spend a relatively small percentage of time in user-mode, with about 20% of time being the smallest for the social network and about 40% of the time in the drone swarm application. The authors ascribe this behavior to the usage of applications like MongoDB and memcached, which spend a lot of their time in the kernel to handle interrupts or processing TCP packets. The large number of cycles spent in libraries is explained by the fact that microservices optimize for speed of development and therefore use a lot of existing libraries. Next, the authors examined the ratio of time spent

in application processing in relation to the time spent in network processing. This metric is compared between the microservice and monolith version of the social network, showing that the microservice version spends a significantly higher percentage of time in network processing, while still having a lower total tail latency. For the other services the percentage of time spent in network processing was compared between low and high load situations, showing that at high load network processing becomes a more pronounced factor of tail latency, up to 50% of time being spent for it in the drone swarm and social network services. This constitutes more than a tripling as compared to the low load situation in the social network case. Following from this, the authors analyze the potential for acceleration of network processing with FPGAs that are used to offload the TCP stack onto. They show that this can improve the end-to-end tail latency between 43% and 220%.

Next, the authors examined the implications of using a microservice architecture on cluster management. They posit that microservices complicate cluster management, because dependencies between the services can introduce backpressure effects, that lead to system-wide hotspots. These hotspots are hard to identify and solve by current cluster managers, because the microservice that is responsible for the delay might not even be saturated itself. Modern multi-tiered architectures are so connected and complicated that it is very hard to automatically identify the right microservices to be scaled out. This also causes the problem that microservice architectures need longer to recover even in the presence of autoscaling mechanisms.

Concerning application and programming framework applications, the authors first studied the per-microservice latency at high and low load to see whether certain microservices are more prone to create the discussed hotspots. The authors concluded that there is a significant difference in all microservices between the distribution of time spent in each microservice between high and low loads, dependent on the choice of programming language and auxiliary services like the back-end database or the front-end load-balancer. However, it is not possible to make any general statements about where to look for bottlenecks even if similar microservices are used in different applications. This is complicated additionally by the fact that these bottlenecks change depending on the load level. The authors also evaluated the use of serverless frameworks regarding their price and performance. Serverless frameworks are well-suited for services with irregular activity where it is not cost efficient to have long running instances. Serverless also improves performance in embarrassingly parallel services, that profit from a big amount of resources. However, they have the disadvantage that applications have to be (re-)programmed to interface with the cloud provider's serverless framework. The author's compared the latency and cost of their applications running in classic containers and in serverless frameworks in the Amazon Cloud with real load. The cost of running in the serverless framework is almost an order of magnitude lower than the containerized solution, while there is almost no latency overhead on average. However, the performance variability was significantly bigger, because functions can be placed anywhere in the datacenter causing unpredictable network latencies. The serverless frameworks also have the advantage that they react a lot more flexibly to increased load compared to the containerized instance with an autoscaling mechanism.

Lastly, the authors examined the effects that occur when systems are scaled out to a large scale with hundreds of users. In their experiments, the authors observed the occurence of large-scale cascading hotspots that are caused by misconfigurations in some of their services. In the end these backed up the entire system. This was only solved by employing rate limiting, affecting user experience by dropping random requests. The authors also measured the influence of unevenly distributed traffic from their users, simulating the situation that 5% of the users are responsible for 30% of the requests in most real-world applications. They observed that when less than 20% of users are responsible for the majority of requests, goodput is almost zero. Lastly, they analyzed the sensitivity of performance to slow servers in the cluster. For large clusters with over 100 servers, when 1% of servers is slowed down, the goodput is already reduced to almost zero, because it is very likely that a server is hosting a microservice on the critical path. This problem does not occur for monolithic architectures since a single slow server only impacts the performance of one replica. In general, the end-to-end performance of a microservice-based system is more impacted by a single slow server the more complex an application's microservice graph is.