ABSTRACT

This is a case study on the benefits of distributing an application for resilience versus consolidation and vertical scaling for increased performance. There are different setups and use cases in the world of distribute systems. Each has different needs. As a user grows, they may want to ensure resilience and increase the available resources by distributing, but how much is overkill? Perhaps it would be more efficient to improve existing machines rather than adding more machines? This paper aims to explore use cases for these scenarios and understand the pros and cons of each approach under certain conditions.

BODY

To conduct testing, a program was developed to stress-test the machine on which it runs. The technology utilized is a Java application, Docker, and Kubernetes. The Java applications makes use of the Java BigDecimal. These are infinite-precision number types. When the program is executed, it discovers how many threads are available on the current machine. It then spins up a process on each of the available threads. This process adds two BigDecimal numbers together repeatedly, with the numbers growing in size until hitting a specified target. Unlike primitive Java types such as ints or doubles, adding two BigDecimal numbers together is expensive and results in using up all available clock cycles on the current thread when the numbers are added repeatedly. Since this process is being run on each available thread, the entire CPU is being operated at 100% usage for a set amount of work. By timing the completion of this work, it can be evaluated how efficiently this work was completed when comparing various setups against one another. This is especially true in the context of a distributed system where processes can run on the same threads in a pod, different pods on a replica, different replicas, and where different replicas may or may not execute on different machines.

For all tests unless otherwise specified (there is one exception), all tests were run on one physical machine. A virtual machine (VM) was then created on this physical machine to host the Kubernetes cluster, with the VM having a fixed amount available of resources. From here, various Kubernetes clusters were created and tested to see how they performed. There are some notes to make about Kubernetes before discussing these tests. First, Kubernetes has a minimum memory requirement of 20 MiB. Anything less than this will result in a failure to create the application used for testing. It will continually attempt to restart and recreate the process, failing each time. Second, this application was developed using an iterative process rather than recursive process. This is because if recursion was chosen, it could not be clearly defined how to involve memory in these tests. In Kubernetes, if a stack overflow is reached, the application is immediately shut down and then restarts, cycling as mentioned above. Scaling memory would not do anything meaningful here because it would only exist in the space of either: there is not enough memory to run the application, or there is enough memory and having the additional bandwidth provides no functional benefit.

Third, as mentioned this program operates iteratively. After some internal testing it was found that the general maximum benefit for expanding memory is approximately 100MiB. This program is written in Java, and thus has garbage collection. Meaning whenever memory is used and more is needed, Java will work to free up additional memory that is in use but not currently needed. It was found from testing that there was no tangible benefit to scaling memory. To explain, expanding the memory capacity on the pod will simply delay the initial time at which Java begins its garbage collection. However, once the garbage collection happens and continues to happen in a long-running program, there will not be a meaningful difference in execution time. Only, if an application has enough memory to run all necessary processes without garbage collection and then after, during some downtime does cleanup, would one see a benefit. However, this would depend on the use case for the application being utilized. The time where expanding the memory will be useful is where there exists an application that is frequently hitting its memory threshold and/or is either consistently getting recycled or has garbage collection running at all times and thus has a slowness factor applied to the application. Even in these scenarios though, it may be better to analyze the application to understand and resolve high memory usage issues instead of simply increasing the memory limit.

A test was conducted in an available cloud environment. It cannot be said whether or not these processes ran on machines that were the same physical machine. Only that the processes each ran on a separate VM each with a standardized amount of available VM resources. Because of this, conjectures cannot be made about physical overhead involved in running this test. Instead, it provides a realistic scenario, showcasing the variance and deviation of distributing an application in cloud where it is possible that processes are running on different machines. Clearly, as indicated by the results in figure x below, there is a wide variance of approximately 20%. This implies that with the many small factors that cannot be controlled in a cloud environment, one should expect their performance to vary request over request for setups where each application is running on a separate machine. While this data does trend to follow a standard deviation, it does show it is still easily possible to see a request that takes 25% longer than another identical request.