Your Very Nice Project Title

Gangmuk Lim
University of Illinois at Urbana-Champaign
{gangmuk2}@illinois.edu

Abstract

Many of modern large systems are composed of a group of individual components. Each component is doing their own dedicated duty based on what it observes from the environment. There are different programming models but we will focus on one where each component keeps checking on the status of the system and tries to make the system converge on a certain desirable status. There is no a single global controller who has the perfect knowledge of the entire system and guarantees that it will not fall into undesirable status. Not just because they are such a big and complex system but there are some particular reasons facilitating the system to be unintentionally faulty. First, multiple components often interact each other and control the same part of the system. Inherently it is possible that they behave contradictorily each other. Seocnd, each component is often developed by multiple people, multiple teams and even multiple organizations. Each team cannot understand how other parts of the system functions in detail. It is almost infeasible and not practical for each component to perfectly take into consideration other parts of the system. This paper did case study to understand the failures involving multiple components. We pick Kubernetes container orchestration system as a representative example to dive into more specific failure cases. We analyzed 10 different failure cases which were reported in Kubernetes github issues, Kubecon (Kubernetes conference), and blog posts.

1 Introduction

The introduction of your awsome 523 project.

2 Background

Kubernetes

Controller Kubelet / Deployment / Scheduler / HPA / Descheduler / api-server / etcd

Reconciliation Monitoring system Distributed tracing e.g., Jaegur, Prometheus, ...

3 Case study

Summarize failure cases + common patterns of failure cases

Detailed examples

Description

Some plots

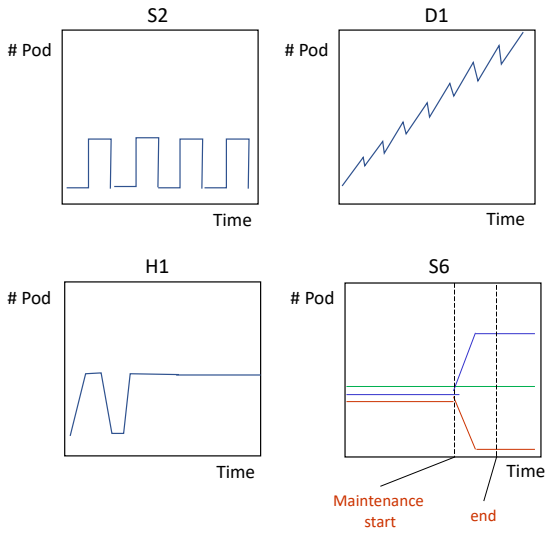

Figure 1. a nice plot

4 Related Work

Testing

Chaos engineering

Model checking

Formal verification

Statistical model The related work of your project [1].

5 Discussion

discussion

modeling / verification /

6 Conclusion

This project is awesome.

7 Metadata

The presentation of the project can be found at:

https://zoom/cloud/link/

The code/data of the project can be found at:

https://github.com/you/repo

References

[1] Dijkstra, E. W. The Structure of the "THE" Multiprogramming System. In *Proceedings of the 1st ACM Symposium on Operating System Principles (SOSP'67)* (Oct. 1967).

Case	Categories	Controllers	Properties	Behaviors
D1(R)	Conflicted config	Deployment + Kubelet	Liveness	Scheduling and evicting pods infinitely
H1(R)	Lack of context	HPA + App CPU changes	Safety	HPA is agnostic to app
H2(R)	Conflicted config	HPA + Deployment	Safety	Sub-optimal scaling behavior
H3(R)	Imperfect knowledge	HPA + Node reachability	Safety	Semantically wrong avg CPU util (reachability vs healthiness)
S1	Conflicted config	Scheduler + Descheduler	Liveness	High utilization(scheduler) <-> Low utilization(descheduler)
S2(R)	Conflicted config	Scheduler + Descheduler	Liveness	Deployment preference <-> Violation in maxSkew
S3(R)	Conflicted config	Scheduler	Liveness	Two pod spread constraints are conflicted each other
S5	Conflicted config	Scheduler	Safety	Pods are scheduled to one node, because of lopsided preference
S6(R)	Lack of feature	Scheduler	Safety	Scheduler is not able to adjust skewed placement
S7(R)	Lack of context	Scheduler + Kubelet	Liveness	Scheduler includes NotReady node for maxSkew calculation