Paragon : QoS-Aware Scheduling for Heterogeneous Datacenters

Introduction and Abstract

1. QoS guarantees provided by cloud platforms is violated due to interference between workloads and sub-optimal resource allocation.
2. Existing solutions to mitigate this problem are computationally intensive, and cannot be applied to a wide variety of applications
3. Paragon is an online and scalable datacenter (DC) scheduler that is heterogeneity and interference-aware.
4. Instead of profiling each application in detail, paragon leverages information about the application based on application history.
5. It uses collaborative filtering techniques to classify unknown workloads with respect to heterogeneity and interference. (Greedy scheduling to minimize interference and maximise server utilisation).
6. The goal of cloud applications is to schedule the stream of incoming applications on available servers that achieves high resource efficiency and fast execution, by enabling scaling at a low cost.
7. This is tough as cloud services have to accommodate a diverse set of workloads, and there is no a priori information about workload characteristics.
8. Heterogeneity occurs because servers are gradually provisioned and replaced, causing different mixtures of server technologies at the same time.
9. Interference is a result of co-scheduling multiple workloads on a single server to increase utilisation and achieve better cost efficiency, where co-scheduled applications may interfere negatively in the presence of shared caches, memory channels, storage and networking devices.
10. A scheduler that is both interference and heterogeneity-oblivious and schedules applications to least-loaded servers is even worse, causing some workloads to crash due to resource exhaustion on the server.
11. Present day schedulers use techniques that cannot scale to large DCs that receive varieties of workloads.
12. Paragon is an online and scalable datacenter scheduler that is heterogeneity and interference aware.
13. Paragon's classification engine exploits existing data from previously scheduler applications and offline training and requires only a minimal signal about a new workload.
14. It uses SVD to perform collaborative filtering and identify similarities in applications' between incoming and previously scheduled workloads.
15. Once an incoming application is classified, a greedy scheduler assigns it to a server that is the best possible match in terms of platform and minimum negative interference between all co-scheduled workloads.

16. The high accuracy of classification negates the effects of greediness, and since the classification is not based merely on empirical observations, there are strong guarantees on accuracy and strict bounds on overhead.

Classification methods for Heterogeneity

1. For high speed and accurate classification we need to know
    a. How fast an application will run on each of the server configs available
    b. How much interference it can tolerate from other workloads
    c. How much interference it will generate itself
2. The system leverages information it already knows about applications it has seen to express the new workload as a combination of known applications
3. In the utility matrix, the rows represent the applications and the columns represent the server configurations, and the ratings represent the normalised application performance on each server configuration.
4. Hidden similarities between applications can be measured by SVD, and these can be filtered on the basis on how much of an impact they would have on the application.
5. The performance was measured based on application types
    a. Single-threaded workloads : Use instructions per second as the initial performance metric
    b. Multithreaded workloads : The presence of spin-locks and other synchronization methods are deceiving. So polling low overhead performance counters is done, to detect changes to the register file and weight out of the IPS computation such execution statements.

Classification methods for Interference

1. There are two types of interferences paragon considers
    a. Interference an application can tolerate from pre-existing workloads
    b. Interference an application causes on the workloads
2. Interference is detected due to contention on shared resources and a score is assigned to the sensitivity of the application to the type of interface.
3. To derive these sensitivity scores, microbenchmarks are used, and tuned up progressively to measure the intensity.
4. Sources of interference (SoI) are identified and microbenchmarks are created for these.
5. Applications are classified for interference tolerated and caused, using twice the process of collaborative filtering used for classifying for heterogeneity.
6. Two utility matrices have applications as rows and SoIs as columns, and the elements are the sensitivity scores to the microbenchmarks.

Putting it all together

1. Paragon requires ~1 minute on two SCs to classify applications for heterogeneity, and another two short runs against two microbenchmarks on a high end SC for interference classification.
2. Running for 1 minute provides some signal on the same new workload without incurring significant overhead.
3. This provides low and tight error bounds on the accuracy of classification, statistical guarantees on the quality of colocation candidates and detailed characterization of system behavior.
4. Scheduler is independent, and hence is scalable and computationally efficient.
5. They also do not introduce significant training and decision overheads to enable exact complexity evaluation.

Link to paper : https://web.stanford.edu/~kozyraki/publications/2013.paragon.asplos.pdf