Poster: Utility-driven fair throughput allocation with latency guarantees over a shared resource pool

Giorgos Kelantonakis, Fallia Kourou, Marina Bitsaki and Kostas Magoutis
Computer Science Department, University of Crete and FORTH-ICS, Heraklion, Greece
{kelanto,kourou,marina,magoutis}@csd.uoc.gr

***Abstract*—In this poster we describe a two-level architecture for achieving quality-of-service (QoS) guarantees for latency-sensitive applications over shared resource pools. We utilize a *higher-level controller* to achieve fair sharing of aggregate system throughput using utility functions expressing each application's goals. A *lower-level scheduler* simultaneously achieves explicit per-application latency targets in concurrent access to a shared pool of resources. The lower-level scheduler isolates each application from others that may exceed their load specifications (average throughput and/or maximum burst size) as decided by the upper-level controller. A distinctive feature of system is that controller allocations are *advisory*, namely an application may decide to exceed them hoping to leverage excess capacity from reduced demand of other applications. Such speculative policies on the side of applications are key to benefiting from spare capacity, if available. They cannot however steal another application's fair share as the scheduler protects compliant applications from violating their latency targets under overload conditions. Our system makes applications aware of their latency metrics at any point in time, so that they can decide if they can afford risking latency violations. We implement and evaluate a preliminary prototype system, to demonstrate its effectiveness in fairly regulating per-client throughput while providing latency guarantees in experiments with synthetically generated workloads. Our results demonstrate that our system is effective in automatically regulating application request rates and in achieving latency targets under workload variations.**

I. Introduction

Shared resource pools, such as consolidated storage arrays, large core-count multiprocessors, etc., are prevalent in data-center environments due to their simpler management and statistical multiplexing benefits. Concurrent access to such resource pools by multiple applications raises the issue of how to ensure fair sharing of resources, isolating well-behaving applications from others that may be hoarding resources, and differentiating between applications under resource shortage. Applications often express their performance goals either via explicit metrics, such as a specific level of throughput and/or latency, or via utility functions expressing the value assigned by the application owner to different performance levels.

II. Design and implementation

Design. In our system, applications specify their latency targets (e.g., <150ms) and their appreciation for throughput (or equivalently, their willingness to pay for it) at their latency target through their utility function. The system achieves fair sharing of resources using the utilitarian criterion, which aims to maximize social welfare, i.e., allocate each application as high throughput as possible maximizing aggregate utility.

Our system features a Controller that is aware of applications (maintains session state for them) and communicates with them via a two-way API. Applications make their characteristics (throughput utility, latency target, maximum burst size) known to the Controller, and expect that the Controller will notify them of an arrival-rate allocation, once per control period. The Controller periodically adjusts the arrival-rate allocations based on the observed system status to effectively manage resource utilization. The assumption that utilization measurements for the resource pool are available allows the Controller to aim for controlling an aggregate metric (U) rather than several individual metrics (per-application latency targets). An important distinctive feature of our system is that applications do not promise to limit themselves to the Controller-reported allocation but may individually decide to exceed them if they determine that they are ahead of their latency goals, as a way to leverage spare capacity (if they have demand for it). This however does not come at the expense of other applications. For performance isolation, our system utilizes a lower-level scheduler that takes as input per-flow load specifications (request rate, maximum burst size, latency target) and schedules requests in a way that will achieve latency targets as soon as the overall load is feasible.

Implementation. Our implementation follows staged event-driven architecture (SEDA) principles and consists of a number of separate modules. Each module/stage is served by a separate thread, and the entire system is run as a single process. This minimizes the latency incurred in information sharing and module communication. More details will be provided on the poster.

III. Future work

Our future work focuses on the scaling of the scheduler and execution modules, while also abstracting the execution module so that it can manage different kinds of resources (e.g. threads, VMs, containers etc). Our long-term goal is to demonstrate that the benefits of this QoS architecture carry on to fully distributed applications and shared resource pools. We thankfully acknowledge support by the Greek Research Technology Development and Innovation Action "RESEARCH-CREATE- INNOVATE", Operational Programme on Competitiveness, Entrepreneurship and Innovation (ΕΠΑνΕΚ) 2014-2020, Grant T2EΔK-02848 (SmartCityBus).