The Linux Scheduler: a Decade of Wasted Cores, EuroSys 2016

(1) Summary

Operating systems designers considered scheduling to be a solved problem. However, as hardware system get complex, scheduling gets bug-prone. Because performance bugs are hard to detect, bugs lasted for years.

This paper detects 4 big bugs using their own tools(sanity checker, scheduler visualization) which were hard to detect before and fix them

First, by adopting minimum loads comparison. between nodes solve group imbalance between multiple application with different thread count.

Second, constructing scheduling group in perspective of the core that arise balancing enable balancing between nodes that are 2-hops apart

Third, waking up a thread on longest idle core prevent system from waking up a thread in overloaded core

Lastly, fixing incorrect update of the number of scheduling domain.

Fixing those bugs significantly improved the system performance.

Name	Description	Kernel version	Impacted applications	Maximum measured performance impact
Group Imbalance	When launching multiple applications with dif ferent thread counts, some CPUs are idle	2.6.38+	All	13×
Scheduling Group Construction	No load balancing between nodes that are 2-h ops apart	3.9+	All	27×
Overload-on Wakeup	Threads wake up on overloaded cores while s ome other cores are idle	2.6.32+	Applications	22%
Missing Scheduling Domains	The load is not balanced between NUMA nodes	3.19+	that sleep or wait	138×

(2) Pros & Cons

strong - Fixing those bugs significantly improved the system performance.

weak - Do not consider power consumption which is important for system. Idling CPUs may not degradate performance that hard while saving energy consumptions

(3) Ideas

Need more in-depth analysis on total system including power consumption

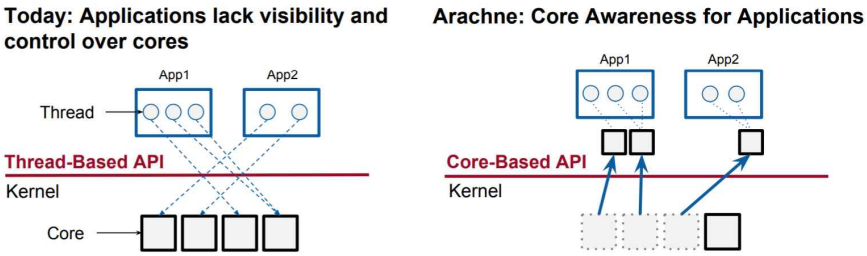
Arachne: Core-Aware Thread Management, OSDI 2018

(1) Summary

Achieving low latency in the data center applications end up lowering core utilization which means low throughput. It is too slow to have one kernel thread handling thread requests. Arachne try to multiplex requests across long-lived kernel threads.

As sharing causes competition for cores, those kernel threads should be distributed per cores.

Also for cache affinity, it is better for applications to have its dedicated cores. To do so applications should know and have control over cores. So Arachne give applications knowledge and control over cores, and move thread management to user-space (application)



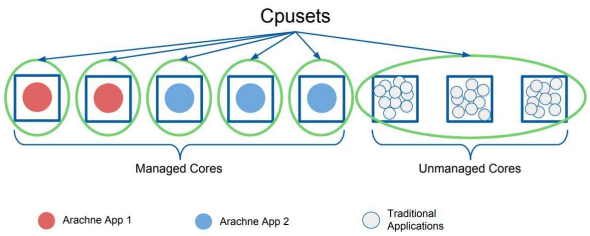
(2) Pros & Cons

Strong

By giving applications direct control to CPU resources, Arachne shows better combination of latency and throughput and efficient thread implementation.

Weak

Favoring Arachne apps(by allocating cores as they wants) may lead the relative slow down on traditional apps.



Application developer should know how to apply the Arachne on their program and implementing it would take a lot of time.

(3) Ideas

The paper does not mention about NUMA system. Adding strategies for managing cores in NUMA system would make this paper more potent.