Core-Aware Scheduling: Balancing Application Parallelism with Core Availability

Henry Qin
Advisor: John Ousterhout

Febuary 2, 2016

PLATFORMLAB

Introduction

Motivation: Inefficient core and thread management

- Hard to get high throughput in low latency services
- Difficult to match application parallelism to available cores.

Proposal: Core-Aware Scheduling

- Thread scheduling moves to user level
- Kernel allocates cores to applications

Outline

Motivation

Proposal for Core-Aware Scheduling

Related Work

Current Status

Request for Feedback

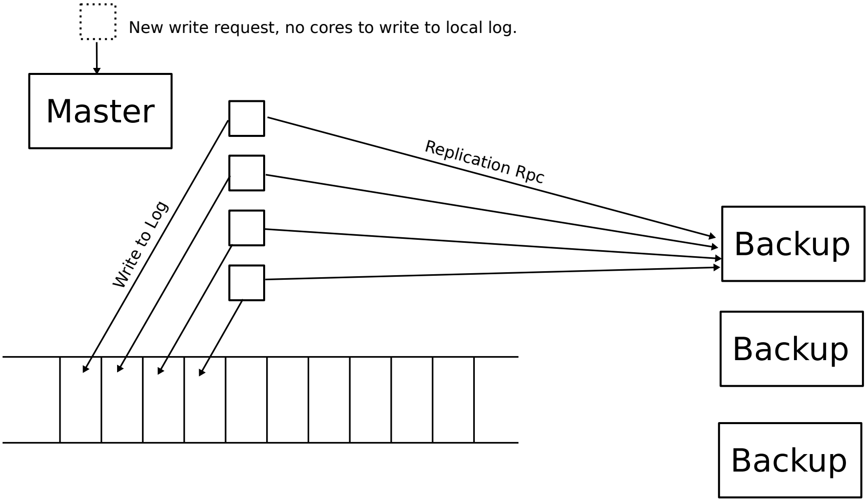
A Throughput Problem

RAMCloud write requests must make replication requests to backup servers, and wait for their return.

RAMCloud uses polling to avoid expensive kernel thread switches and and kernel bypass to avoid system calls.

When the master runs out of CPU cores it must cease processing requests.

Core Exhaustion Bottleneck



What happens under load?

Backups are slower to respond, since they coexist with masters.

Write requests wait even longer for backups, spinning cores for even longer.

Match application parallelism to available cores

Application servers can have many threads running such as log cleaners, worker threads, and failure detection threads.

We want to neither overcommit nor undercommit cores.

Overcommit cores ==> undesirable kernel multiplexing because there are multiple kernel threads per core

Under commit cores ==> idle cores.

When the log cleaner needs to run, we would like to scale down the number of worker threads so that we do not exceed available cores.

Core-Aware Scheduling: Kernel Core Allocator

Kernel scheduler class which allocates cores to applications on request.

In general, kernel never preempts a thread running on the cores it has allocated to the process.

Allow kernel to safely multiplex latency-sensitive applications with CPU-bound batch jobs.

Latency-sensitive applications can request only as many cores as they need, and give up cores when they no longer need it.

Core-Aware Scheduling: Userland Scheduler

Fast context switches enable practical core multiplexing in a low-latency system.

Manage thread priorities and parallelism level based on application-specified policies.

User-level scheduler requests dedicated cores from the OS, and always knows exactly how many cores it has.

Preempted Questions

How will you handle system calls for blocking IO?

Why is thread pinning insufficient?

Related Work

Scheduler Activations inspired this work but it not sufficiently core-aware because the kernel makes too many scheduling decisions.

Related Work

Scheduler Activations inspired this work but it not sufficiently core-aware because the kernel makes too many scheduling decisions.

Linux cgroups do not allow support the dedicated allocation of specific cores.

Related Work

Scheduler Activations inspired this work but it not sufficiently core-aware because the kernel makes too many scheduling decisions.

Linux cgroups do not allow support the dedicated allocation of specific cores.

Cappricio does not support multicore.

Related Work

Scheduler Activations inspired this work but it not sufficiently core-aware because the kernel makes too many scheduling decisions.

Linux cgroups do not allow support the dedicated allocation of specific cores.

Cappricio does not support multicore.

Go does not address the core allocation problem; no mechanism to communicate with kernel for dedicated cores.

Related Work

Scheduler Activations inspired this work but it not sufficiently core-aware because the kernel makes too many scheduling decisions.

Linux cgroups do not allow support the dedicated allocation of specific cores.

Cappricio does not support multicore.

Go does not address the core allocation problem; no mechanism to communicate with kernel for dedicated cores.

Cilk requires user threads to be non-blocking.

Related Work

Scheduler Activations inspired this work but it not sufficiently core-aware because the kernel makes too many scheduling decisions.

Linux cgroups do not allow support the dedicated allocation of specific cores.

Cappricio does not support multicore.

Go does not address the core allocation problem; no mechanism to communicate with kernel for dedicated cores.

Cilk requires user threads to be non-blocking.

OpenMP supports neither core allocation nor explicit management of thread scheduling.

Current Status

Implemented a simple user-level dispatcher.

Measured a single direction context switch with no cache pollution at 9 ns on an Intel(R) Xeon(R) CPU X3470 @ 2.93GHz

Request for Feedback

Request for Feedback

Do you know of a threading system that solves these problems of core allocation and fast context switching practically and cleanly?

Request for Feedback

Do you know of a threading system that solves these problems of core allocation and fast context switching practically and cleanly?

Have you ever measured the core utilization over short time intervals (ms and s) on your large-scale systems?

Request for Feedback

Do you know of a threading system that solves these problems of core allocation and fast context switching practically and cleanly?

Have you ever measured the core utilization over short time intervals (ms and s) on your large-scale systems?

Do you have dedicated hardware or shared machines?

Request for Feedback

Do you know of a threading system that solves these problems of core allocation and fast context switching practically and cleanly?

Have you ever measured the core utilization over short time intervals (ms and s) on your large-scale systems?

Do you have dedicated hardware or shared machines?

How do you decide on the number of OS threads for an application?

Request for Feedback

Do you know of a threading system that solves these problems of core allocation and fast context switching practically and cleanly?

Have you ever measured the core utilization over short time intervals (ms and s) on your large-scale systems?

Do you have dedicated hardware or shared machines?

How do you decide on the number of OS threads for an application?

What is the relationship between this number and the number of cores on the machine?

Thank You!

If we did not talk at the poster session, please find me at the reception!

Send mail to hq6@cs.stanford.edu

Questions?