Go-RealTime: Knowledge and Control of Time in High Level Programming Language

Award # CNS-1329755 (UCLA), CNS-1329644 (CMU), CNS-1329644 (UCSD), and CNS-1329650 (UCSB)
Type: Frontier; Start Date: June 2014

Zhou Fang (UCSD)
Co-Authors: Sean Hamilton, Hao Zhuang, Rajesh Gupta (UCSD)

Introduction

Go-RealTime is a new multiprocessor real-time framework, based on Go language. It integrates key functionalities of programming **time sensitive applications** into language runtime: **Quality of Time (QoT)**, **Real-Time Scheduler** and **Parallel Programming.**

This work provides the software/programming language level support for the **RoseLine QoT Stack**.

APIs

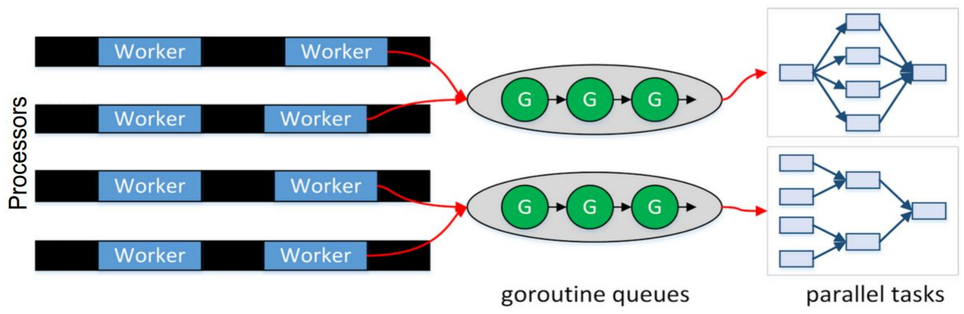
We import OS APIs (thread scheduler, CPU affinity and timer) into Go, modify Go runtime to enable **direct goroutine switch**, and build external Go packages.

Type	Method	Description
Resource	*SetSched(thread, policy, priority)* *BindCPU(CPU)* *SetTimerFd(file_descriptor, time)*	Linux thread scheduler Processor affinity Linux timer
Scheduling	*GoSwitch(goroutine)*	Switch to a goroutine
Task	*NewWorker(Nw)* *task.SetTimeSpec(ts, tp, td, tr, QoT)* *task.Run(func, arg)*	Create *Nw* workers Set timing specification Start a RT task

Design

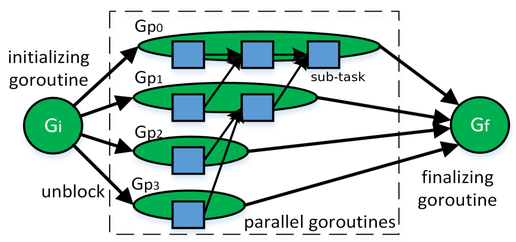
System Model

Worker is the abstraction of thread with a processor resource reservation. Goroutines are sorted in a few priority queues. A sequential task is associated with a goroutine.

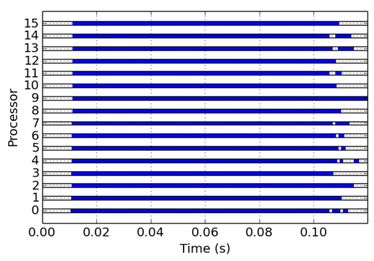


Parallel Programming

Go-RealTime models a **parallel task** as a **DAG of sub-tasks**, executed on a group of parallel goroutines, scheduled by global EDF algorithm.



Synchronization of parallel goroutines via channels



Parallel dynamic programming on 16 cores

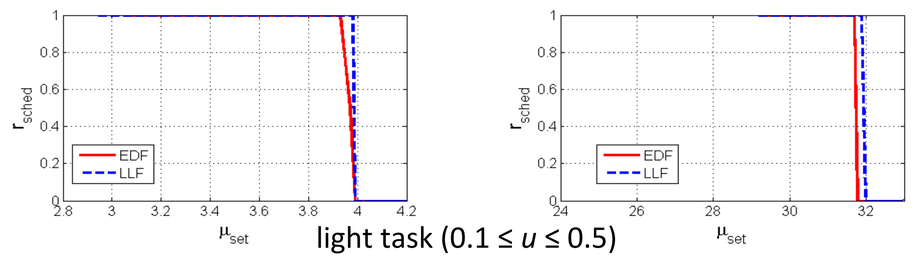
Features

- **Goroutine** is the concurrency unit on which sequential and parallel tasks are running, controlled by Real-Time Schedulers.
- **OS thread** is the unit of **processor resource reservation**. Linux thread schedulers (FIFO, RR, etc.) are imported into Go.
- **Task** describes a real-time program with a **timing specification:** {start ***ts***, period ***tp***, deadline ***td***, worst case budget ***tr***, utilization ***u = tr/tp***, **QoT**}
- Adaptive timing accuracy of asynchronous events (e.g. timer, task switching) upon **QoT** requirements of tasks.
- **Parallel tasks** are programmed as Directed Acyclic Graph (DAG) of sub-tasks and executed on multiprocessor.
- Different types of tasks are scheduled by the suitable algorithm: Earliest Deadline First (EDF), Least Laxity First (LLF).
- The total processor resources are partitioned dynamically among all schedulers according to utilization of tasks.

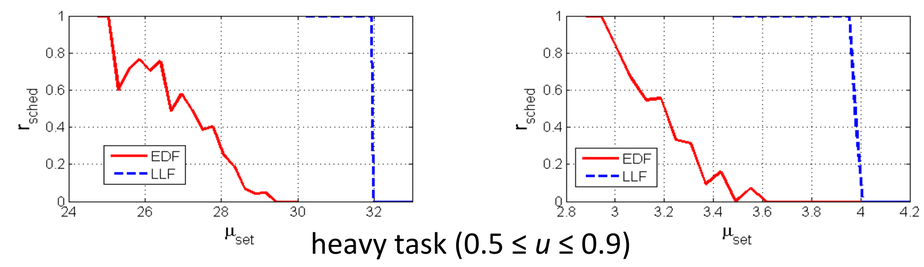
Real-Time Scheduler

Algorithm

EDF/LLF schedulers are implemented for sequential tasks. For **LLF**, Go-RealTime compares laxities of the running task with the head task in queue via *check_async* method. A switch is allowed only after ***tllf*** **(default 10ms)** since the previous.



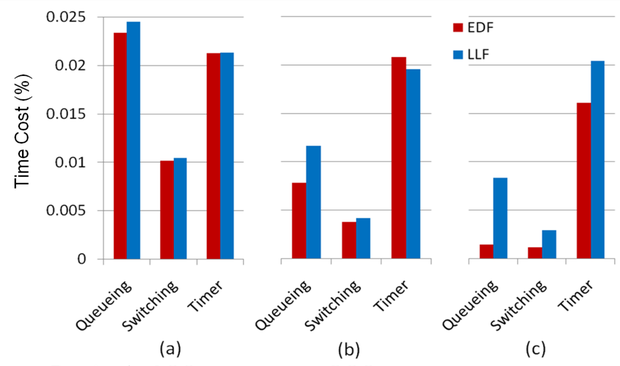
light task ($0.1 \le u \le 0.5$)



heavy task ($0.5 \le u \le 0.9$)

System Cost

- **Queueing** cost: time consumed by scheduler code
- **Switching** cost: direct cost of goroutine switch
- **Timer** cost: cost of operations on timer queue such as insertion
- **Indirect** cost: cache pollution due to switching



Period: 100ms ≤ tp ≤ 300ms

(a) ultra light task ($0.01 \le u \le 0.1$); (b) light task; (c) heavy task

Asynchronous Events and QoT

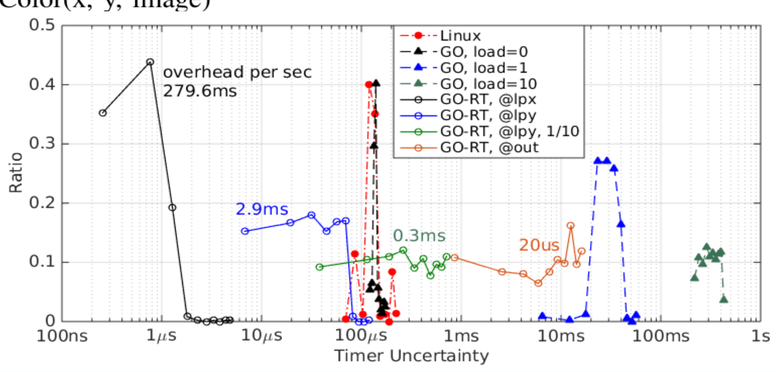
Go-RealTime handles **asynchronous events** via '***check_async***' method. It checks all asynchronous events (timeout, message, etc.) and responds to the events which should have occurred.

Source code of Go-RealTime programs is instrumented with *check_async* calls by Go parser library. The locations of calls are adjusted ahead according to measured timing profile. The calls are dynamically enabled/disabled to satisfy **QoT** requirement at runtime.

```
1: @out: check_async()
2: for loop over y coordinate do
3:     @lpy: check_async()
4:     for loop over x coordinate do
5:         @lpx: check_async()
6:         SetGrayColor(x, y, image)
7:     end for
8: end for
```

Instrumented *convert_gray* function

Timer accuracy of Linux, Go and Go-RealTime



Scheduling Algorithm Change

In the example, when the total utilization increases, the scheduler changes from EDF to LLF in order to avoid deadline miss.