Migrate when necessary: toward partitioned reclaiming for soft real-time tasks

RTNS'2015

H. ZAHAF[1], G. LIPARI[1],L. ABENI[2]

[1] CRIStAL, Lille 1 University
[2] Scuola Superiore Sant'Anna

Grenoble, 04/10/2017

Plan

Introduction

Our Contributions

Results and discussions

Conclusion

Soft Real-time

- Undesirible to miss deadlines but not crucial
- Modern operating systems can serve this kind of applications

- Examples
  - Multimedia coding/decoding/transmission
  - Networking, cellular operations,
  - ...
- Execution time is not know apriori
- Tasks may have a unperidictible behavior.

Consequence

- Non-garantee to respect all deadlines
- Tasks with long execution times may monopolize the processor, and then increase deadline miss rate.
- → *special* scheduling techniques

Resource Reservation and reclaiming

Temporal isolation

- Ressource reservation allow to have temporal isolation:
  - To each task is assigned a **service time** “budget” Q_i in each period of time P
  - Tasks running out their budgets must not condition other’s tasks execution.

Reclaiming

- What about tasks that runs less then their budgets?
  - Need a reclaiming policy to take benifit from the *unused* budget.

Reclaiming: Singlecore and multicore

Single-core

- Several algorithms exist for single-core ressource reservation and reclaiming:
    - Grub, CBS, CASH, $\cdots$

Multicore support

- Multicore techniques are: global, partitioned or semipartitioned.
- Several extension of GRUB and CBS has been proposed to support multicore (Global Parallel Grub, Sequential Parallel Grub [], M-CACH[]).

Global vs Partitioned ??

Partitioned:

- Each core is has it own ready-queue
- Signle-core reservation & reclaiming strategy

Pros

- Easy to implement: No migration
- a good exploitation of the platform

Cons

- No-inter core reclaiming (under utilization of resources)

Global:

- One ready Queue for all cores
- The m highest priority jobs are run

Pros

- Implicit inter core reclaiming

Cons

- Hard to implement,
- A big overhead due to the high number of migrations

Problem formulation

Our Solution

Is in the middle:

- Partition tasks to cores
- allow jobs to migrate to other cores to seek for extra-budget & tasks to migrate to have “a better” QoS.

We have:

- a set of tasks, each task is a set of jobs and :
  - A server, an abstract scheduling entity
  - The server contains necessary informations to schedule tasks:
    - Budget, Period, $\cdots$
- a set of identical cores

Goal

Parition tasks and allow jobs to reclaim the unused bandwith on all the cores (inter-core reclaiming).

Plan

Introduction

Our Contributions

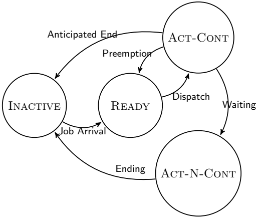
Results and discussions

Conclusion

Single-Core Grub: Recall

Grub server uses 3 variables :

- Bandwidth $u_i = \frac{Q_i}{p_i}$
- Server's deadline d_i. The task with the earliest d_i is executed.



- V_i The virtual time, is updated when the task is in active-contending state: $\mathsf{V}_i(t+\Delta t) \leftarrow \mathsf{V}_i(t) + \frac{\mathsf{U}^a}{\mathsf{u}_i}\Delta t, \mathsf{U}^a = \sum_{\mathcal{S}_i \in \{\text{ACT-CONT}\} \cup \{\text{ACT-N-CONT}\}} \mathsf{u}_i$.
- if $\mathsf{V}_i > \mathsf{d}_i$, the deadline is postponed as $\mathsf{d}_i = \mathsf{V}_i + \mathsf{P}_i(*)$
  - Task budget is exhausted.
  - Priority loss.

Grub with Job migration

Idea

- Before postponing the deadline, the job may seek to find extra-budget with the same urgency.
    - Keep the temporal isolation on the destination core
    - Migrate only when necessary (Sufficient budget is ensured in destination core)

- Each server is characterized by an extra variable : u_i^m
- u_i^m is the needed bandwidth to “eventually” complete before deadline on the destination core.

When job migration is allowed

- A job is eligible for migration at time t_0 if :

$$\mathrm{V}_i(t_0) \geq \mathrm{d}_i \tag{1}$$

$$\mathrm{d}_i > t_0 \tag{2}$$

- Selection of destination core
  - The task must fit in destination core : $\mathrm{u}_i^m + \mathrm{U}_{j'}^m(t) + \mathrm{U}_{j'} \leq 1$
  - Ensure that with the current maxium load, the task can run *sufficient* execution time: $\mathrm{u}_i^m(\mathrm{d}_i - t)/(\mathrm{u}_i^m + U_m^a) > \epsilon$
- if this condition is not respected, migrating utilization can be reduced

Job Migration

Before migration:

- New temporary server is created and initialized to READY
- Virtual time is set to t_0(migration moment)
- Server deadline is kept the same
- Migration flag is set to limit the migrations of the same job.

At the end of migration:

- the task returns immediately to its original processor at job completion.
- Migration flag is set to false

Task migration

In the case of open systems:

- New tasks arrival → load unbalance → performance degradation

Solution

- *permanently* migrate the task to improve its quality of service.
- Permanent task migration can be performed at the end of the execution of its active job.
- A task permanant migration → all its future jobs will start executing on the new processor.
- Permanent migration is triggered if $miss\_rate^{now} > miss\_rate^{max}$

Permanent migrations

Migrations are:

- immidiate, if the migration core verifies:

$$\mathsf{u}_i \leq 1 - \mathsf{U}_j - \mathsf{U}_j^m$$

- delayed, if previous condition not verified.
  1. Select a core that verifies: $\mathsf{u}_i \leq 1 - \mathsf{U}_j$
  2. Lock migrations to the selected core
  3. Trigger the migration when the previous condition is verified
- aborted, if non of the preivous conditions is verified.

Plan

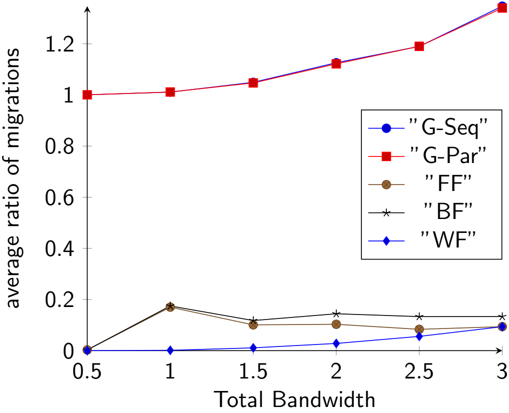
Introduction

Our Contributions

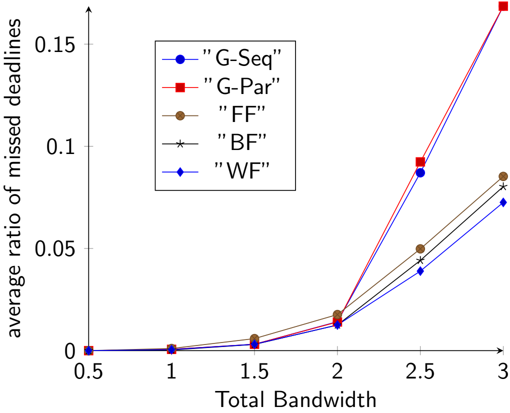
Results and discussions

Conclusion

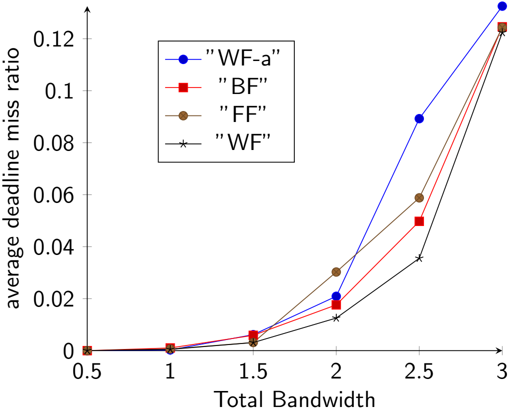
Job migration Vs Global scheduling: #Mig./job



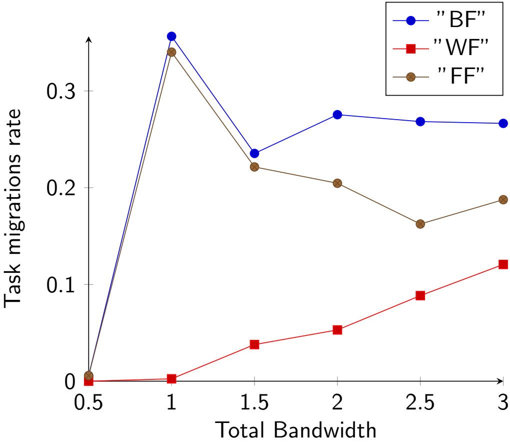
Job migration Vs Global scheduling: Dead-miss/job



Job & task migration Vs Global scheduling: #Mis/job

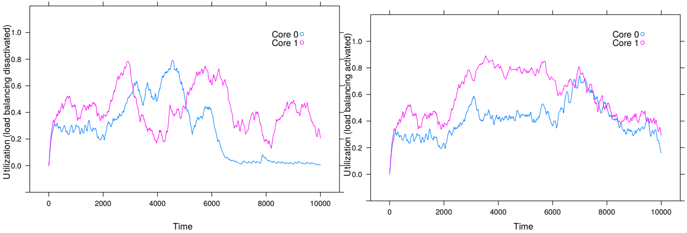


Job & task migration: #Mig./job

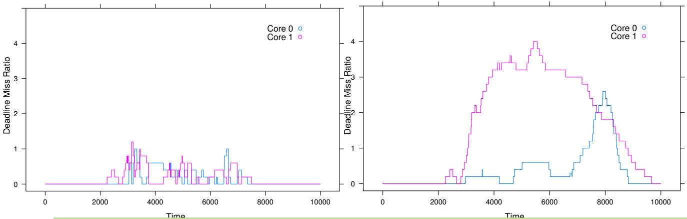


On-line task arrival/leaving

Active utilization



Deadline miss ratio:



Plan

Introduction

Our Contributions

Results and discussions

Conclusion

Conclusion

- We proposed:
  - heuristics to partition a set of soft real-tasks to a set of identical cores
  - The execution time of each job is not know apriori
  - Tasks may enter and leave the system dynamically
- We allow:
  - Jobs to migrate to reclaim extra-budget
  - Tasks to migrate to

Futur Work

- Implement the partitioned grub with job & task migration on Linux Kernel.
- Extend the current partitioned grub to hererogeneous architectures: ARM-bigLITTLE
- Complement the current version with DVFS and DPM to reduce the energy consumption.