TECHNISCHE UNIVERSITÄT DRESDEN

LUND UNIVERSITY

Models@Run.Time Workshop 2015

USING REFERENCE ATTRIBUTE GRAMMAR-CONTROLLED REWRITING FOR ENERGY AUTO-TUNING

Christoff Bürger Department of Computer Science, Lund University, Sweden
Johannes Mey Software Technology Group, TU Dresden, Germany
René Schöne Software Technology Group, TU Dresden, Germany
Sven Karol Chair for Compiler Construction, TU Dresden, Germany
Daniel Langer Software Technology Group, TU Dresden, Germany

Ottawa, September 29, 2015

Presentation Overview

Our **new idea**: Use Reference Attribute Grammars and rewriting for runtime models.

We use

- a **Reference Attribute Grammar (RAG)**
- to create and modify a **runtime model**
- of batch process execution on a compute cluster and
- use **attributes** and **RAG-controlled rewrites** to schedule the system's tasks
- in an **energy-optimized** way.

Outline

Case Study

Solution Background (*RACR*)

Our Solution

Evaluation and Outlook

A Case Study: Scheduling Batch Processes

of Wikipedia Indexing Tasks

Very simple case study to show use of RAGs for runtime model

Task: indexing of text chunks (taken from Wikipedia)

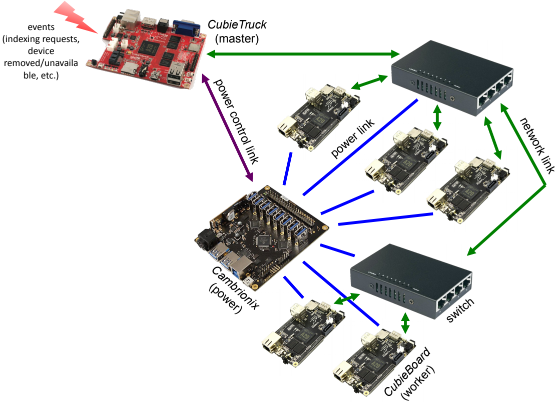
- processing time predictable (proportional to chunk size)
- requests arrive interactive (occur randomly)
- requests have deadline

Energy Optimization: Minimize energy consumption of the indexing system

- System is network of (embedded) computers
- Computers (and connecting switches) can be turned off to save energy

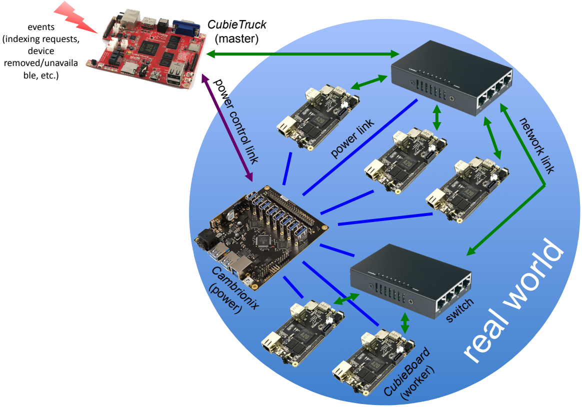
A Case Study: Scheduling Batch Processes

of Wikipedia Indexing Tasks



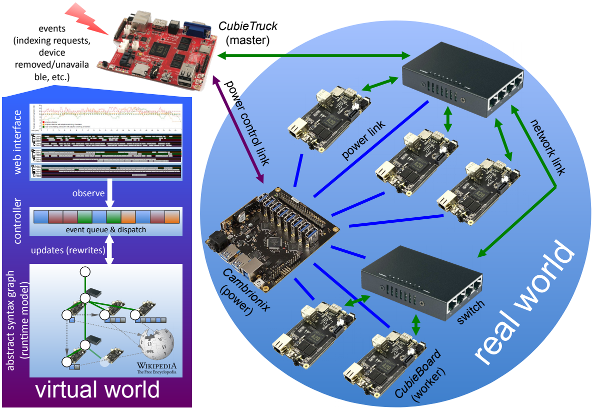
A Case Study: Scheduling Batch Processes

of Wikipedia Indexing Tasks



A Case Study: Scheduling Batch Processes

of Wikipedia Indexing Tasks



Solution Background

RACR - Reference Attrribute Grammar-Controlled Rewriting

RACR is . . .

- a Reference Attribute Grammar (RAG) system
  - declarative semantics
  - lazy, incremental evaluation
- for RAG-controlled rewriting
  - advanced AST manipulation

Solution Background

RACR - Reference Attribute Grammar-Controlled Rewriting

RAG-controlled rewriting = RAGs + graph rewriting

- reference attribute grammar for declarative **analyses**
  - reference attributes induce sematic overlay graph on top of *abstract sytax tree (AST)*
  - enables deduction *and* analyses of graph structure

  → deduced, memoized *abstract syntax graph (ASG)*
- graph rewriting for ASG **transformations**
  - left hand: ASG pattern (ASTs connected via reference attributes)
  - right hand: manipulations on matched underlying AST

  → ASG changes with AST (updated by RAG)
- seamless combination:
  - use analyses to deduce rewrites
  - rewrites automatically update analyses

  → incremental

Solution Background

RACR - Reference Attribute Grammar-Controlled Rewriting

The Implementation: RACR

- reference implementation of RAG-controlled rewriting in *Scheme* R6RS[1]

RACR contains API for:

- ASG schema definition (AST schema + attribution)
- ASG querying (AST + attributes)
- rewriting:
  - imperative **and/or** RAG-controlled **and/or** fixpoint
  - primitive **and/or** pattern-based
  - **... in any combination!**

`https://github.com/christoff-buerger/racr`

Solution Background

RACR - Reference Attribute Grammar-Controlled Rewriting

The Implementation: RACR

- reference implementation of RAG-controlled rewriting in *Scheme* R6RS[1]

RACR contains API for:

- ASG schema definition (AST schema + attribution)
- ASG querying (AST + attributes)
- rewriting:
  - imperative **and/or** RAG-controlled **and/or** fixpoint
  - primitive **and/or** pattern-based
  - **... in any combination!**

`https://github.com/christoff-buerger/racr`

[1] Don't panic! C bindings and .NET bindings are available.

Our Solution

The Grammar

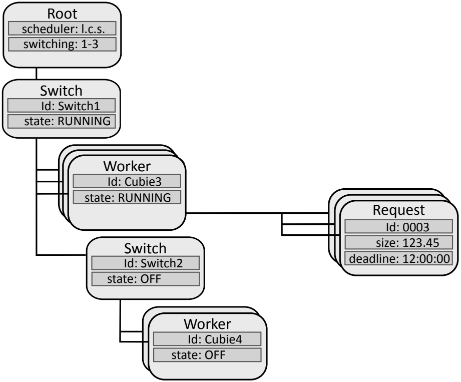
Grammar is encoded in Scheme symbols

- production rule: left side -> right side
- upper case: nonterminals
- lower case: terminals
- repetition (*), inheritance (:)

```
(ast-rule 'Root->scheduler-backupworkers-CompositeWorker)
(ast-rule 'AbstractWorker->id-state-timestamp)
(ast-rule 'CompositeWorker:AbstractWorker->AbstractWorker*)
(ast-rule 'Switch:CompositeWorker->)
(ast-rule 'Worker:AbstractWorker->devicetype-Request*<Queue)
(ast-rule 'Request->id-size-deadline-dispatchtime)
```

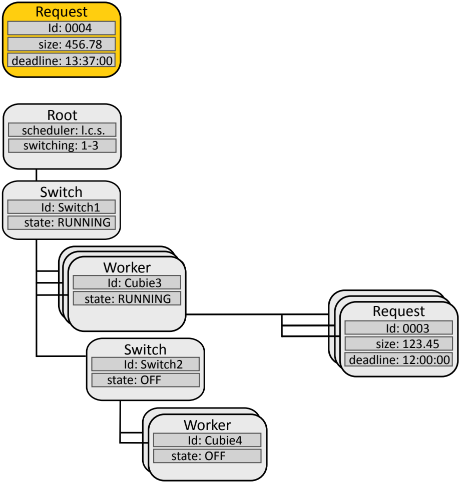
Our Solution

Example AST



Our Solution

Scheduling a Request



Our Solution

Scheduling by Rewriting

Scheduling a new task: rewriting the AST

- insert a new Request node at the right position

```
(rewrite-insert
  (ast-child 'Queue worker) ;list-node to insert into
  index ;position of insertion
  (create-ast spec 'Request (list id size deadline #f)))
```

Our Solution

Attribute-controlled Scheduling

Where to put the new Request?

- evaluate attribute **schedule** to find insertion position
- result is worker and position in worker’s queue
- Attribute depends on terminal **scheduler**

  → scheduler can be exchanged at runtime!

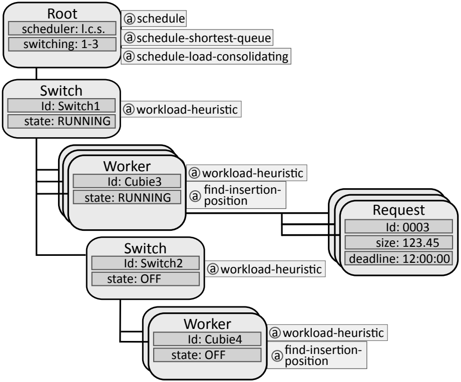
```
(ag-rule schedule
  (Root (lambda (n time work-id load-size deadline)
    (att-value (ast-child ’scheduler n) n time work-id
                load-size deadline))))
```

Our Solution

Two schedulers implemented:

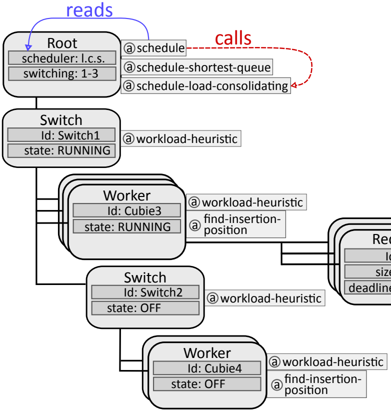
schedule-shortest-queue simple scheduler inserting in shortest queue of any worker

schedule-load-consolidating inserts request in fullest queue while ensuring deadline is kept



Our Solution

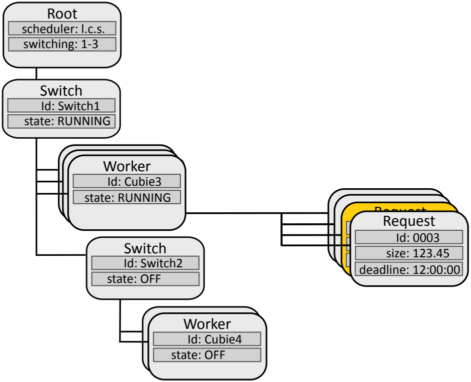
Attribute evaluation



Our Solution

Scheduling a Request

The resulting AST



Our Solution

Saving Energy

Required workers are computed with attributes

- adaptation strategy regularly computes *how many* and *which* workers to switch on or off
- interactive system requires backup workers
- amount of backup workers and adaptation parameters described in AST

Saving energy by switching off workers:

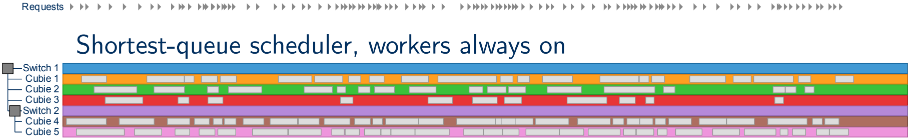
- Try to minimize amount of idle workers
- use adaptation strategy
- use load-consolidating scheduler to minimize required workers

Evaluation

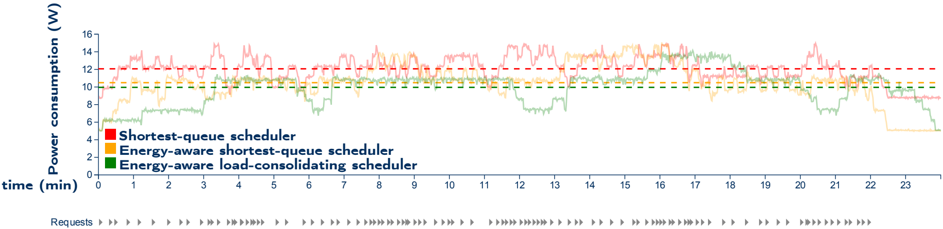
Test setup for measuring energy consumption

- graphical interface to show system state and consumed power
- Scenario generator to run controlled workloads with different settings

Evaluation

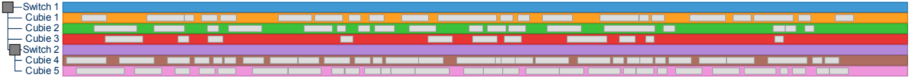

Requests
Shortest-queue scheduler, workers always on
Switch 1
Cubie 1
Cubie 2
Cubie 3
Switch 2
Cubie 4
Cubie 5

Evaluation

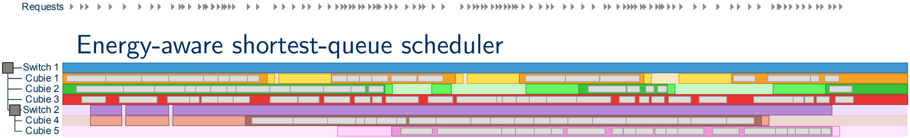


🟥 Shortest-queue scheduler
🟧 Energy-aware shortest-queue scheduler
🟩 Energy-aware load-consolidating scheduler

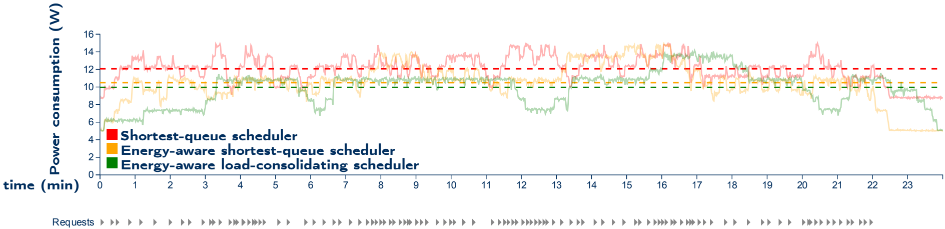
Shortest-queue scheduler, workers always on



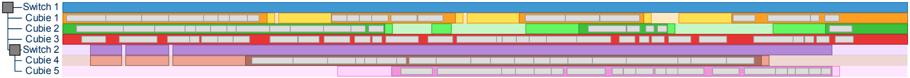
Evaluation


Requests
Energy-aware shortest-queue scheduler
Switch 1
Cubie 1
Cubie 2
Cubie 3
Switch 2
Cubie 4
Cubie 5

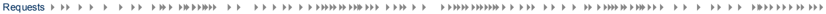
Evaluation



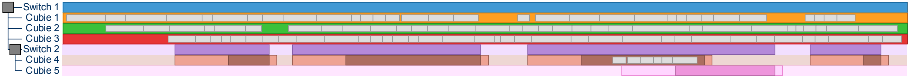
Energy-aware shortest-queue scheduler



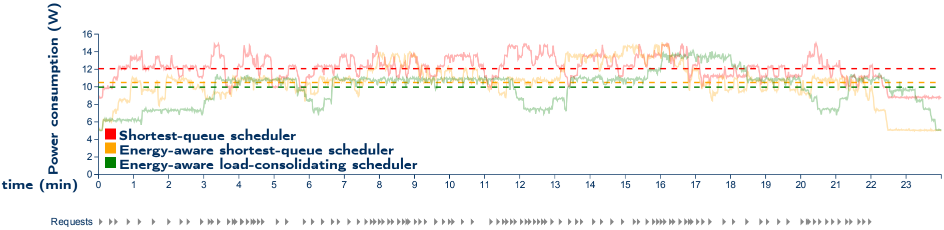
Evaluation



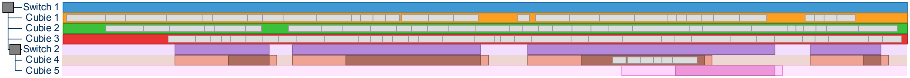
Energy-aware load-consolidating scheduler



Evaluation



Energy-aware load-consolidating scheduler



Evaluation

Properties of the Solution

- **scalable**: incremental evaluation ensures only necessary attributes are re-evaluated after system change
- **adaptive**: ASG structure can be modified at runtime, schedulers and parameters can be switched
- **fault-tolerant**: system can handle device failures

Evaluation

Results

- Energy-aware shortest queue scheduler saves 13.1% compared to regular shortest-queue scheduler
- Energy-aware load-consolidating scheduler saves 17.5% compared to regular shortest-queue scheduler
- load-consolidating scheduler increases amount of request that can be scheduled

Outlook

Next steps

- heterogeneous architecture
- more interesting network structure
- simulate large systems

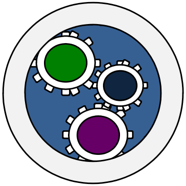
- more case studies for RACR for runtime models

Conclusion

Benefits of RAG-Controlled Rewriting

interactive …

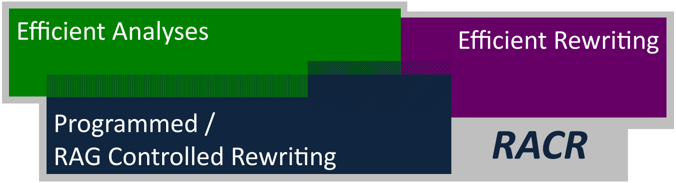
runtime models



incremental reasoning

model transformation

…



Efficient Analyses

Efficient Rewriting

Programmed / RAG Controlled Rewriting

RACR