Homework index

1 Bizarre scheduling idea ........................... 36
2 Reading assignment .............................. 37
Goals for lecture

• Sensor networks
• Finish overview of scheduling algorithms
• Mixing off-line and on-line
• Design a scheduling algorithm: DCP
  – Will initially focus on static scheduling
• Useful properties of some off-line schedulers
Lab two?

• Everybody able to finish?
• Any problems to warn classmates about?
• 18 motes should be arriving tomorrow
  – No equipment sign-out required for next motes lab
• Linux vs. Windows development environments
Sensor networks

- Gather information over wide region
- Frequently no infrastructure
- Battery-powered, wireless common
- Battery lifespan of central concern
Low-power sensor networks

• Power consumption central concern in design

• Processor?

• Wireless protocol?

• OS design?
Low-power sensor networks

- Power consumption central concern in design
- Processor?
  - RISC μ-controllers common
- Wireless protocol?
- OS design?
Low-power sensor networks

• Power consumption central concern in design

• Processor?
  – RISC $\mu$-controllers common

• Wireless protocol?
  – Low data-rate, simple: Proprietary, Zigbee

• OS design?
Low-power sensor networks

• Power consumption central concern in design

• Processor?
  – RISC $\mu$-controllers common

• Wireless protocol?
  – Low data-rate, simple: Proprietary, Zigbee

• OS design?
  – Static, eliminate context switches, compile-time analysis
Low-power sensor networks

- Power consumption central concern in design
- Runtime environment?
- Language?
Low-power sensor networks

- Power consumption central concern in design
- Runtime environment?
  - Avoid unnecessary dynamism
- Language?
Low-power sensor networks

• Power consumption central concern in design

• Runtime environment?
  – Avoid unnecessary dynamism

• Language?
  – Compile-time analysis of everything practical
Multi-rate tricks

• Contract deadline
  – Usually safe

• Contract period
  – Sometimes safe

• Consequences?
Scheduling methods

- Clock
- Weighted round-robin
- List scheduling
- Priority
  - EDF, LST
  - Slack
  - Multiple costs
Scheduling methods

- MILP
- Force-directed
- Frame-based
- PSGA
Linear programming

• Minimize a linear equation subject to linear constraints
  – In $P$

• Mixed integer linear programming: One or more variables discrete
  – NP-complete

• Many good solvers exist

• Don’t rebuild the wheel
MILP scheduling

(P) the set of tasks

(t_{\text{max}}) maximum time

(start(p,t)) 1 if task p starts at time t, 0 otherwise

D the set of execution delays

E the set of precedence constraints

\[ t_{\text{start}}(p) = \sum_{t=0}^{t_{\text{max}}} t \cdot \text{start}(p,t) \] the start time of p
MILP scheduling

Each task has a unique start time

\[ \forall p \in P, \sum_{t=0}^{t_{\text{max}}} \text{start}(p, t) = 1 \]

Each task must satisfy its precedence constraints and timing delays

\[ \forall \{p_i, p_j\} \in E, \sum_{t=0}^{t_{\text{max}}} t_{\text{start}}(p_i) \geq t_{\text{start}}(p_j) + d_j \]

Other constraints may exist

- Resource constraints
- Communication delay constraints
MILP scheduling

• Too slow for large instances of \( \text{NP-complete} \) scheduling problems
• Numerous optimization algorithms may be used for scheduling
• List scheduling is one popular solution
• Integrated solution to allocation/assignment/scheduling problem possible
• Performance problems exist for this technique
Force directed scheduling


- Calculate EST and LST of each node

- Determine the force on each vertex at each time-step

- Force: Increase in probabilistic concurrency
  - Self force
  - Predecessor force
  - Successor force
Self force

\( F_i \) all slots in time frame for \( i \)

\( F'_i \) all slots in new time frame for \( i \)

\( D_t \) probability density (sum) for slot \( t \)

\( \delta D_t \) change in density (sum) for slot \( t \) resulting from scheduling

self force

\[ A = \sum_{t \in F_a} D_t \cdot \delta D_t \]
Predecessor and successor forces

**pred** all predecessors of node under consideration

**succ** all successors of node under consideration

**predecessor force**

\[
B = \sum_{b \in \text{pred}} \sum_{t \in F_b} D_t \cdot \delta D_t
\]

**successor force**

\[
C = \sum_{c \in \text{succ}} \sum_{t \in F_c} D_t \cdot \delta D_t
\]
total force: $A + B + C$

- Schedule operation and time slot with minimal total force
  - Then recompute forces and schedule the next operation
- Attempt to balance concurrency during scheduling
Force directed scheduling
Force directed scheduling

task duration

EST

LST
Force directed scheduling
Force directed scheduling

probabilistic concurrency
Force directed scheduling

probabilistic concurrency
Force directed scheduling

• Limitations?

• What classes of problems may this be used on?
Implementation: Frame-based scheduling

- Break schedule into (usually fixed) frames
- Large enough to hold a long job
  - Avoid preemption
- Evenly divide hyperperiod
- Scheduler makes changes at frame start
- Network flow formulation for frame-based scheduling
- Could this be used for on-line scheduling?
Problem space genetic algorithm

- Let’s finish off-line scheduling algorithm examples on a bizarre example
- Use conventional scheduling algorithm
- Transform problem instance
- Solve
- Validate
- Evolve transformations
Examples: Mixing on-line and off-line

- Book mixes off-line and on-line with little warning
- Be careful, actually different problem domains
- However, can be used together
- Superloop (cyclic executive) with non-critical tasks
- Slack stealing
- Processor-based partitioning
Problem: Vehicle routing

- Low-price, slow, ARM-based system
- Long-term shortest path computation
- Greedy path calculation algorithm available, non-preemptable
- Don’t make the user wait
  - Short-term next turn calculation
- 200 ms timer available
Examples: Mixing on-line and off-line

- Slack stealing
- Processor-based partitioning
Scheduling summary

• Scheduling is a huge area

• This lecture only introduced the problem and potential solutions

• Some scheduling problems are easy

• Most useful scheduling problems are hard
  – Committing to decisions makes problems hard: Lookahead required
  – Interdependence between tasks and processors makes problems hard
  – On-line scheduling next Tuesday
Bizarre scheduling idea

• Scheduling and validity checking algorithms considered so far operate in time domain

• This is a somewhat strange idea

• Think about it and tell/email me if you have any thoughts on it

• Could one very quickly generate a high-quality real-time off-line multi-rate periodic schedule by operating in the frequency domain?

• If not, why not?

• What if the deadlines were soft?
Reading assignment


• Read Chapter 7