e-Merge-ANT: November 2000

Kestrel Institute
Stephen Fitzpatrick, Cordell Green & Lambert Meertens
http://ants.kestrel.edu/
ANTs PI Meeting, Charleston, SC, 28-30 November 2000

Outline

- Status
- Anytime scheduler with anytime graph coloring
- Results using simulator
- Comments on challenge problem
Status

**Current Achievements**

- Informal Architecture Specifications
- Formal Resource & Task Specifications
- Anytime Scheduling Algorithms
- Java Code
- Communication & Tracking Skeleton
- RadSim (& Hardware)

**Plans**

- +Formalize
- +Synthesis
- Experiments in Dynamics
- +Analysis of Dynamics
- +Scheduler Visualizer

+Formalize

+Synthesis
Distributed, Anytime Rescheduling

An algorithm for scheduling radar nodes
- meet mission objectives (track targets)
- reduce resource consumption

Operational requirements
- scaleable: complexity independent of number of nodes
- distributed: tolerant of communication latency
- real-time: responds quickly enough to track targets effectively
- robust: degrades gracefully as, e.g., communication or hardware fails
- incremental: schedules ongoing, dynamic tasks
Distributed, Local Repair Algorithm

Define a distributed set of scheduling processes
  – each scheduling process is responsible for some set of local resources
  – schedules for two resources are in conflict if they together cause a constraint violation

Define neighborhoods
  – two resources are neighbors if they interact
    • e.g., there is some constraint that relates the two resources

Define local quality metric on schedules
  – e.g., number of conflicts at a node
    • requires neighbors to inform each other about schedules
Each scheduling process follows an iterative procedure:

- it locally optimizes its own schedule with respect to its neighbors’ schedules
  - e.g., to accommodate new tasks & to reduce its conflicts with its neighbors
- and then informs its neighbors of its new schedule

Diagram showing iterative process of local repair with improvement.
Communication Latency/Synchronization

- Each scheduling process optimizes its schedule wrt its neighbors’ schedules
  - optimization is based on information at hand
  - neighbors may have changed schedules
  - an optimization wrt neighbors’ old schedules may be a degradation wrt actual current schedules
  - result is poor convergence

Need to synchronize update & exchange of schedules

![Diagram showing communication latency and synchronization]

- unscheduled tasks (%)
- time
- asynchronous
- sequential

Local repair without improvement
Totally Sequential Synchronization?

- Extreme case: totally sequential operation across system
  - ensures every change is made with up-to-date information
  ⇒ no change produces a worse schedule

- BUT, sequential operation is not scaleable
  - at any given time, only one scheduling process throughout the entire system may update its schedule
  - (and communicate the new schedule to its neighbors)
  - Complexity $\propto$ number of nodes
Graph Coloring for Synchronization

- Use graph coloring to achieve sufficient synchronization
  - nodes of the (undirected) graph are scheduling processes
  - two graph nodes have a connecting edge if they interact
  - color the nodes so that no two nodes of the same color have an edge between them

- At any given time, only one color is “active”
  - all of the scheduling processes of that color may update
  - all other scheduling processes must wait

⇒ Interacting processes (neighbors) cannot change schedules simultaneously

- Require number of colors $\ll$ number of nodes
  - number of colors $=$ number of nodes $⇒$ sequential operation
  - number of colors $= 1$ $⇒$ totally parallel operation
Graph Coloring: Complexity of Scheduling

- Number of scheduling processes: $N$
- Minimum number of colors required: $C_{\min}$
- $N/C_{\min}$ scheduling processes can be active simultaneously
  - high degree of parallelism

$\implies$ Complexity independent of size of system

- $C_{\min}$ depends on “interaction topology”
  - at most $C_{\min}$ scheduling processes directly interact
  - non-local task structures/constraints give high $C_{\min}$
    - truly global constraints cause $C_{\min}$ to be equal to $N$
    - indicative of (theoretically) non-scaleable deployment platform
Distributed, Anytime Graph Coloring

How to compute a coloring in a distributed environment?

Apply similar local repair process to graph coloring:
  – a color conflict occurs when two neighboring scheduling processes have the same color
  – each process repeatedly selects that color which (currently) minimizes its conflicts with its neighbors

Need to address convergence of coloring
  – at each stage, use whatever coloring is available to synchronize coloring process
  – even an imperfect coloring reduces the probability of simultaneous changes offsetting each other

Coloring and scheduling proceed simultaneously
  – an imperfect coloring may also be beneficial for the scheduling process
Requirements Met?

❖ Scaleable? Constant complexity
   – complexity is independent of number of nodes

❖ Distributed? Convergence is achieved via coloring
   – a high latency will still slow down the processes
   – it dictates the cycle time

❖ Real-time? A schedule is always available
   – provides real-time framework
   – time bounds affect the quality of schedules

❖ Robust? Each scheduling process operates on information available
   – missing information will degrade schedules due to unresolved resource conflicts
   – but some results will still be available

❖ Incremental? Continually reschedules
Analysis

To date, analysis is of tracking results
- outstanding objective: analysis of scheduling

Example track

Ground truth: times, position vectors, velocity vectors

\[ G = [ t_i \times \vec{g}_i \times \vec{u}_i, i=1.. ] \]

Tracker output: times, position vectors, velocity vectors

\[ R = [ t_k \times \vec{p}_k \times \vec{v}_k, k=1..n_R ] \]

Error vectors (in position)

\[ \vec{e}_k = \vec{p}_k - \text{interpolate}(G, t_k), k=1..n_R \]

Display color \( \sim |\vec{e}_k| \)
- green good - red bad

High-error points due to target being “lost”
- time required to reacquire
Track Display

Kestrel

RadSim Example
Analysis: Overall Performance

- Representative results using simulator
  \[ \text{R.M.S.} = \sqrt{\left(\sum |\vec{e}_k|^2/n_R\right)}, \ k=1 \ldots n_R \]
  \[ = 3.09 \text{ feet} \]

- Average beam usage
  \[ = \text{total beam seconds} / \left(3 \times \text{number of nodes} \times \text{simulation duration}\right) \]
  \[ = 27\% \]

- Communication usage
  \[ = 0.9 \text{ messages per second per node} \]

![Track Error Distribution](image)
Analysis: Track Animation

- Shows ground truth path
- Shows track positions
  - sliding/fading window over actual track positions
  - linear interpolation between positions (with velocity)
  - color coded to show error (linear interpolation)
- Shows tracker’s a priori prediction of target path segment
- Shows radar beam usage
- Implemented in VRML 2.0 for convenience
  - allows control of animation speed, direction
  - pre-defined and user-controlled viewpoints
  - maybe move to Java3D or X3D
Track Animation: Movie

- 90 second pre-rendered movie shown here
- Approximately 5x normal speed
**Analysis: Tracker Grid**

- **Larger sample size**
  - 6000 track points

- **Grid artifact**
  - Track positions show correlation with 1 foot × 1 foot grid used by tracker to compute target locations that best match radar measurements
Comments on Challenge Problem Error Metric

- Error metric discussed on mailing list
  - shortest distance to ground-truth path
    \[ \sqrt{\sum \text{distance}(G, \bar{p}_k)^2/n_R}, \ k=1..n_R \]

- Error metric we used
  - interpolate ground-truth path using track point’s time coordinate
    \[ \sqrt{\sum |\bar{p}_k - \text{interpolate}(G, t_k)|^2/n_R}, \ k=1..n_R \]

- Neither metric takes into account the number of track points
  - a track having just one measurement may score highly

- Proposal: interpolate both ground and track positions to \( n_I \) points evenly spaced over duration of simulation
  - approximated path integral
    \[ \sqrt{\sum |\text{interpolate}(R, t_j) - \text{interpolate}(G, t_j)|^2/ n_I}, \ j=1..n_I \]
    \[ t_j = j \times \text{(simulation duration)}/n_I \]
Summary

- Have produced a slice from specification to code
  - need to refine the specifications
  - and tie them to code using synthesis

- Performance of tracker & scheduler seems reasonable
  - need to try larger systems with multiple targets

- Need further experiments to analyze scheduler performance
  - synthesize family of implementations for experimentation

http://ants.kestrel.edu/
References

VRML 2.0 (a.k.a. VRML 97) http://www.vrml.org/
– open, standardized, plain text format for 3D scene description
– animation described using key frame techniques
  • e.g., time-position coordinates
  • CPU/system speed determines quality of animation (frame rate)
– VRML scene can be viewed using any compliant viewer
  • e.g., plugins for Netscape and Internet Explorer
– good 3D graphics card needed for reasonable frame rate (>8 fps)
– never quite reached critical mass, but some stalwarts remain (e.g., Parallel Graphics, Blaxxun)

X3D http://www.web3d.org/x3d.html
– open format being developed as replacement for VRML 2.0

Java3D http://www.j3d.org/
– open API for 3D scene construction & viewing in Java
– VRML scene can be viewed using stand-alone applications or objects/applets embedded in web pages