Local Rules optimize the Organization of Processes in Networks

Based on “Explicit Design and Adaptation in Self-Construction”
PhD Thesis by Fabian Roth, 2007

Michael Pfeiffer
pfeiffer@igi.tugraz.at

CI Seminar A, 30.01.2008
Limits of computational power on single processors are reached \(\Rightarrow\) parallelization

Efficiently distributing tasks over multiple CPUs with low-bandwidth communication channels is non-trivial

What can we exploit?
- Known properties of task (e.g. sub-processes)
- Known architecture of the network on which we parallelize the task (e.g. grid, client/server,...)
- What if we don’t have this prior information?
Two Examples

- **Nervous system**
  - Single cell: minimal computational capabilities
  - Low communication bandwidth
  - Power comes from high dimensionality and complex wiring

- **Graphical models** (e.g. factor graphs) are similar
  - Simple computations at nodes (sum / max)
  - Simple messages sent from node to node
  - Event-driven update
Goals of this Work

- Arbitrary network topology
  - Even changing topology allowed

- Single, simple processes on network nodes

- Complex computations emerge through interaction of simple units

- Ability to self-construct and self-repair
  - Optimize organization of processes on network
Agenda

- Introduction
- Networks of Distributed Processes
- Routing
- Process Migration in Networks
- Embedding of graphical models in networks
- Learning of connectivity from observation
Self Development of Process Networks

- Abstraction of earlier principles
  - Physical substrate → Computer network
  - Cell → Process
  - Chemical Diffusion → Message passing

- Only local rules guide organization of processes
  - No global observer!

- Processes live on nodes of a network
- Processes send out messages through links of network
Process Model

- **Core**
  - Management of messages and modules

- **Modules**
  - dynamically loaded
  - Addressed messages

- **Message**
  - Source and destination
  - Type and class
  - Message body

- **Modules can serialize themselves**
  - Program code is sent over the network (migration)
Dynamic Routing

- Task: transport message from node to other node
  - Local decision: which channel to use?
  - Value of sending message to destination over a particular channel

- Classical solution: *Bellman-Ford (BF)* routing
  - Global observer computes shortest path and stores lookup tables at each node
  - Basis for most routing schemes used today, e.g. IP-routing
Q-Routing

- Local, adaptive version of BF-routing
- Reinforcement Learning of Q-function
  - Value = estimated time to reach destination
  - $Q_x(y, d)$: at node $x$: value of sending a message to destination node $d$ via neighbor $y$
  - After sending message to $y$, receive update value
    - $Q_y(\hat{z}, d) = \min_{z \in N(y)} Q_y(z, d)$
    - $q_y$ ... delay from $x$ to $y$

$$
\Delta Q_x(y, d) = \eta \left( Q_y(\hat{z}, d) + q_y - Q_x(y, d) \right)
$$
Q-Routing

- After convergence \( Q_x(y, d) = Q_y(\hat{z}, d) + q_y \)
- Learn good routing policy with some exploration

- Dual Reinforcement Q-Routing
  - Faster convergence if learning occurs in both directions
  - Message from source \( s \) contains information how to best reach \( s \) from current node

\[
\Delta Q_y(x, s) = \eta \left( Q_x(\hat{z}, s) + q_x - Q_y(x, s) \right)
\]

- No additional messages needed, slight increase of message size
Age Based Q-routing

- Even if messages diffuse randomly
  - Messages from source \( s \) most likely arrive through a channel that comes from a node close to \( s \)
- Strategy
  - Send message to destination \( d \) via a channel through which most messages from \( d \) arrived
  - Self-enforcing
  - Update at receiver-side, not sender-side
Age-based Update Rule

- Influence of message on Q-value decays with time
  \[ \Delta Q_y(x, s) = \eta \left( \exp(-t_M \alpha) - Q_y(x, s) \right) \]
  - \( t_M \) ... age of message
  - \( \alpha \) ... aging factor

- Values inversely proportional to delay
- Q-values decay without usage
  - Allows changing topologies
  - Best routes are updated more frequently
- Immediate re-biasing of better routes
  \[ Q_y(x, s) = \max(Q_y(x, s), \exp(-t_M \alpha)) \]
  - Immediately remembers shortest latency
Results

- 4 processes in grid-network, communication 1-4 and 2-3
- Limited capacity of edges (congestion)
- One (left) or two (right) bottlenecks join network parts
  - Shortest path alone cannot avoid congestion
• Age-based Q-routing finds shortest paths
• With two bottlenecks traffic is redirected to optimize overall performance
• 1-connector performance equivalent to shortest-path
• 2-connector performance significantly better
• Selfish optimization leads to optimal compound delivery time
Process Migration

- Process may change its position in network to optimize overall information delivery
  - e.g. move closer to sender/receiver
  - Based on previous local routing algorithm

- Value of node $x$ for process $i$
  $$V_i(x) = \max_{y \in N(x)} Q_y(x, i) \approx \exp(-t_i(x)\alpha)$$
  - $t_i(x)$ ... fastest time to reach process $i$ from $x$

- Induces *distance metric* on network related to positions of processes
  - Locally known, can be used to guide *migration* of processes
Migrational Objective Function

- A process needs to optimize latencies to all processes with which it communicates → find optimal position
  - Send out migration queries to neighbors
  - Monte-Carlo movement: always move to better location, stochastically move to worse location

- Objective function

\[ O_t(x) = \sum_{j \in C} \langle l_j \rangle - \sum_{j \in C} \sqrt{\langle (\Delta l_j)^2 \rangle} - \pi n_x \]

\( l = \{\log V_j(x)\} \) ... vector of negative latencies to all connected processes
\( n_x \) ... number of processes on node x (penalty factor \( \pi \))

- Goals:
  1. minimize message ages
  2. minimize scattering of message ages (std. deviation)
  3. avoid gathering of multiple processes on one node
- Process 1 and 2 fixed, 3 can migrate
- Process attempts to be close, but equally distant to all connected processes
Embedding Graphs into Networks

- Graph Node → process
- Graph Edge → communication channels

- Grow a graph from single process/cell and optimally arrange it to network topology
  - Topology may change over time!

- Additional capabilities of graph processes
  - Contain internal description of entire graph
  - Can replicate into two processes
Embedding Example

- Single process tries to connect to graph neighbors
- If this fails → divide
- New process is graph neighbor
- New process behaves exactly in the same manner
- Connections of node 3 shown
- Local rules
  - Self-constructing
  - Self-maintaining
Embedding Example

- Single process tries to connect to graph neighbors
- If this fails → divide
- New process is graph neighbor
- New process behaves exactly in the same manner
- Connections of node 3 shown
- Local rules
  - Self-constructing
  - Self-maintaining
Altered Network Topology

- Interrupt communication
  - New routes are found
  - Processes migrate
  - Self-repair

- Now cut network in two
  - Figure (f)
Altered Network Topology

- Network is cut in two
  - Two sub-graphs re-build the whole graph
  - Two equivalent copies are built
  - Self-repair

- For full graphical model
  - Functional module required
  - Here only structure
Learning of Graphical Models

- Learning a simple world model
- Processes observe some (binary) variables from the world
- Causal connections are modeled as connected processes
  - Causal: $p(x_2=1 \mid x_1=1) = 1$
  - Hidden causes are allowed, e.g. $p(x_1), p(x_3)$

**Directed Acyclic Graph (DAG)**

**Possible World States**
Learning the Model Structure

- Task: Observe states of the world and infer causal relationships
  - $D$ … set of observed data
  - $G$ … graph structure

- Structure learning approaches
  - Maximize likelihood $L(G \mid D) = P(D \mid G)$
  - Bayesian inference, using $P(D \mid G)$ and prior $P(G)$
    - Here: incremental local rule
Causal Learning Rule

- Local update rule:
  \[ P_{i \rightarrow j} = P(x_j | x_i) \approx \frac{f_{ij}}{f_i} \]
  probability of cause \( i \rightarrow j \)

  \[ f_{ij} \rightarrow f_{ij} + x_i \cdot x_j \]
  frequency of joint occurrence

  \[ f_i \rightarrow f_i + x_i \]
  frequency of occurrence of \( x_i \)

- \( x_i, x_j, P_{i \rightarrow j}^{\text{pre}} \) available at node
- bookkeeping for \( f_i \)
- replace \textit{learning rate} \( 1/f_i \) by constant \( \eta \)
Results of Learning Rule

- Plot: $x_{\text{row}}$ caused by $x_{\text{col}}$, learned from 2000 samples
  - Bottom row thresholded
  - Correlation as comparison
- Relation $8 \rightarrow 7$ is learned
  - Only $1/11$ of examples contain $x_7=1$, $x_8=0$
  - $9/11$ suggest $8 \rightarrow 7$
Analysis of Learning Rule

\[
\Delta P_{i \rightarrow j} = \frac{x_i}{f_i} \cdot (x_j - P_{i \rightarrow j}^{pre}) \approx \eta \cdot x_i \cdot (x_j - P_{i \rightarrow j}^{pre})
\]

- Learning rule has equilibrium at true \( P(x_j \mid x_i) \)

\[
\left\langle \Delta P_{i \rightarrow j} \right\rangle_{P(x_j \mid x_i)} = 0
\]

- Variance of update decays with \( f_i^2 \), thus converges to 0

\[
Var(\Delta P_{i \rightarrow j}) = \frac{P(x_i, x_j)}{f_i^2} \left(1 - P(x_j \mid x_i)\right)
\]

  - For causal scenarios replacing \( 1/f_i \) by constant learning rate \( \eta \) accelerates convergence
  - Still, \( \eta \) should be chosen sufficiently small for small variance
Simulation in Network of Processes

- Observable variables (*sensory variables*)
  - Stationary processes that broadcast messages into the network

- *Observables* rebuild the causal graph
  - Moveable and replicating processes
  - Occupy nodes with sensory processes
  - Every sensor requires observer to make variables available to other processes in the network

![Diagram showing network of processes with observable variables s1 and o1, and connections to target process]
Modules for Observer Processes

- Observable replicator:
  - Wait for first broadcast message and specialize to observe this variable
  - Upon receiving other messages replicate and order new process to observe new variable

- Migration Module: optimize process location
- Sensor Module: broadcast variable values

- Causation Module:
  - Use learning rule to learn causations from incoming broadcasts
  - Add target processes if $P_{i \rightarrow j}$ rises above threshold (also remove too weak causal links)
Example

- Sensory variables s1-s9 placed randomly in y-dimension
- Alternate between possible world states every 2 seconds
- Start with single “Mother” – observable
Example

- Initial random specialization of Mother cell to o7
- Replication and migration of observables
- Table of causal links built
Learned Connectivity

After 300 sec simulation

- Indirect causes (e.g. 1→5) are learned
Connectivity from Temporal Correlation

- A consistently before B \(\Rightarrow\) A might cause B
- A consistently after B \(\Rightarrow\) A cannot cause B

- Idea of Spike Timing Dependent Plasticity (STDP)
  - Replace previous rule with STDP
  - Temporal simulation of input
  - Indirect causes not captured
    - Indirect causes need longer to take effect
Learned Connections with STDP

- Almost perfect reconstruction of graph
- No indirect causes
- Connection 6 → 7 incorrectly learned
  - Chains of events (3 → 6 and 3 → 5 → 7) lead to 6 being active one step before 7
Prediction and Novelty

- Predicted events need not be transmitted
  - Only *novelties* need to be sent around
- Suppress broadcasts of sensory variables if activity can be predicted by previous observables
  - Still send signal to connected processes
- Simulation: 85% of events correctly detected as novelties / caused events
Discussion

● Self-organization of processes on arbitrary network topology through local rules
  • Routing and process migration

● Embedding of graph structures

● Local rules for structure learning
  • Learn world model from observation
  • Use model for prediction
Discussion

- True learning architecture
  - Less pre-programmed rules and parameter settings
  - Communication targets learned from data

- Combine structure learning and functionality
  - Can we learn real probabilistic world models?

- Possible uses
  - Structure learning for graphical models
  - Peer-to-peer networks

- Does it relate to neural organization?
  - Processes not single neurons but functional areas