Minerva: A Reinforcement Learning-based Technique for Optimal Scheduling and Bottleneck Detection in Factory Operations

Tara Elizabeth Thomas, Jinkyu Koo, **Somali Chaterji**, and Saurabh Bagchi

Dependable Computing Systems Lab (DCSL)
School of Electrical and Computer Engineering
Purdue University

Funded by GE Center at Purdue, PRIAM (2015-20)

Roadmap

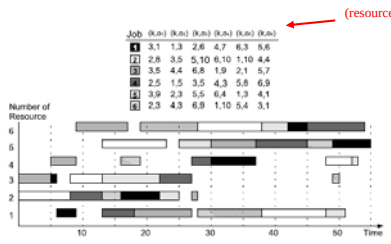
- Motivation
- Background
- Our Solution: Minerva
- Experimental Results
- Conclusion

Motivation

- Simulation-based modeling of factory operations on the rise
  - Improved simulation software
  - Can virtually play with the system configurations without actual impact
  - Data-driven approach based on sensor data for performance improvement
- **Key Idea**: Try to leverage simulation-based modeling to improve performance
  - Optimize scheduling
  - Identify and eliminate bottlenecks

Job Shop Scheduling Problems (JSSP)

- Well studied in literature
  - Creates optimal schedule of jobs to machines
  - Assume finite number of jobs

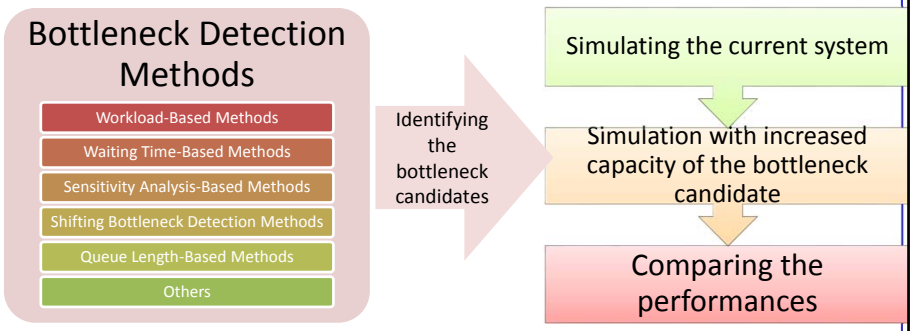


Different from task scheduling in an OS.
- Each job must be processed in a specific order of resources.

- Not very realistic
  - Factories have continuous job arrival
  - Need to adapt scheduling policy based on system state
  - Can extend these models to make them more realistic

Bottleneck Identification

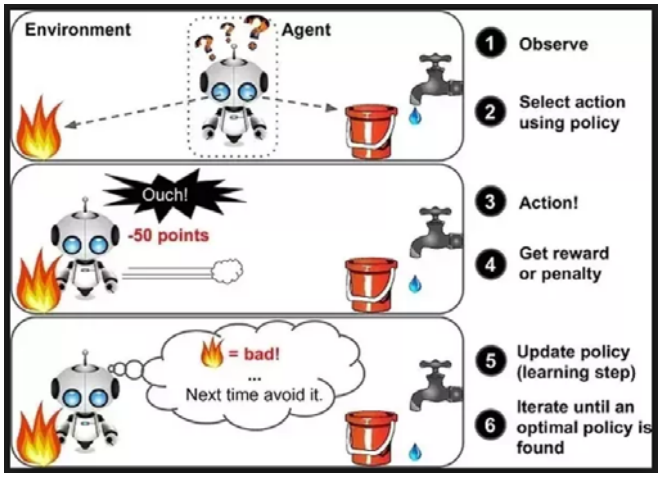
Definition - Bottleneck resource: The resource whose capacity increase results in maximum throughput increase.



Different techniques work well for different scenarios

Background: Reinforcement Learning

- RL makes a software agent learn the optimal action for each system state.



Background: Q- Learning and Approximate Q-Learning

- Can find the optimal action-selection policy for any given Markov Decision Process.
- Calculates Q-function - expected utility if a particular action is chosen in a particular state.
- In realistic models, possible state space is huge
  - Direct application of table based Q-learning algorithms become impractical.
- Approximate Q-learning techniques
  - Q-function is approximated by decision trees or neural networks
  - Deep Q-learning with experience replay - very efficient technique for huge state spaces

PURDUE UNIVERSITY

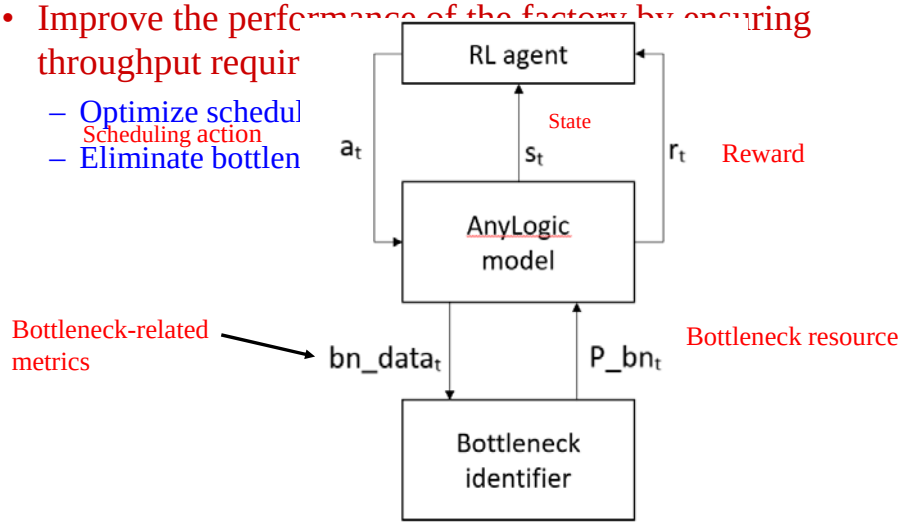
Roadmap

- Motivation
- Background
- Our Solution: Minerva
- Experimental Results
- Conclusion

PURDUE UNIVERSITY

Minerva Overview

- Improve the perfc[illegible] throughput requir
  - Optimize schedul
  - Eliminate bottlen

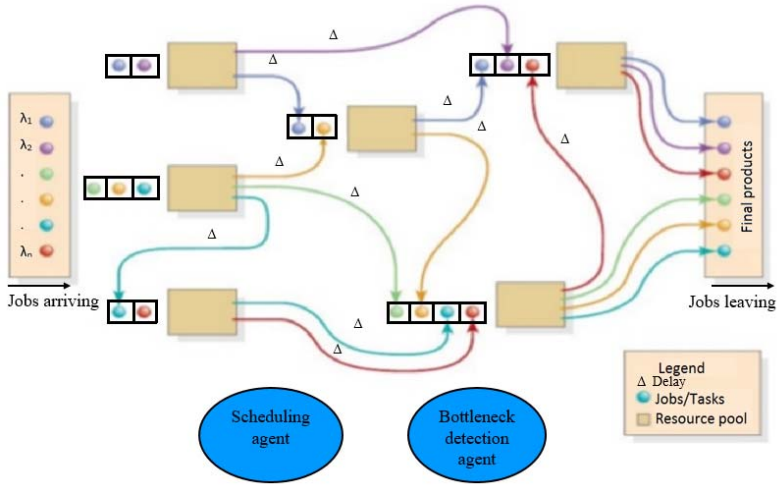


Key Idea

- Embed intelligent agents into the simulation model
  - Learn best scheduling policies based on experience of the agent: Use RL
  - Intelligent agents learn the optimal schedule for each state of the system
  - Once it learns, use it for actual scheduling!
- Use Neural Network to predict bottleneck resource
  - Learn what combinations of metrics give most accurate prediction
  - Increase capacity of bottleneck resource
  - Re-do optimal scheduling
  - When performance measure is met, terminate

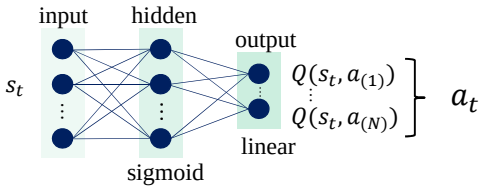
AnyLogic Model:

Overall Distributed Factory System Model

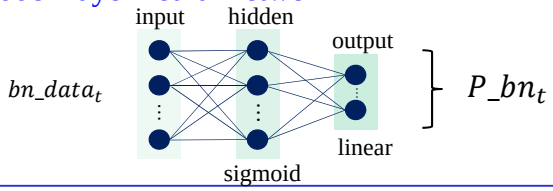


Agent Models

- RL agent
  - 1 hidden layer neural network



- Bottleneck identifier
  - 1 hidden layer neural network



System Model

- Single Markov Decision Process(MDP)
- State of the system at any time *t*, *s(t)* has:
  - Utilization and capacity of each resource pool
  - Number of jobs waiting for each operation
  - Current throughput

Large and dynamically changing state space

  - Continuously arriving stream of jobs, with Poisson distributed arrival rates.
  - Introduces stochasticity into the previously deterministic JSSP model

The RL Scheduling Agent

- Takes decisions on which job to process next based on Q-learning
  - When a resource becomes idle
  - When a new job enters an empty queue of an idle machine
- Action space: Set of integers
  - Represents which job type to be processed at which machine
  - Dynamic
    - There may not be all jobs waiting for the resource at that time instant.
- Reward function:

$$r_t = \begin{cases} k_1 \cdot H_t \text{ if } s_t \neq s_T \\ k_2 \text{ if } s_t = s_T \text{ and } H_t \geq H \\ -k_2 \text{ if } s_t = s_T \text{and } H_t \leq H \end{cases}$$

The Bottleneck Identification Agent

- Detects the bottleneck resource and increases its capacity
  - Assumes single bottleneck at any time *t*
  - The capacity of the bottleneck is increased by 1 unit
- Uses a combination of information from the state space:
  - Queue lengths
  - Average waiting times
  - Utilizations of machines
- Uses neural networks for detection of bottlenecks
  - Pre-trained using random simulation outputs
  - Essentially a classifier that takes above metrics and outputs the resource which is the bottleneck

PURDUE UNIVERSITY

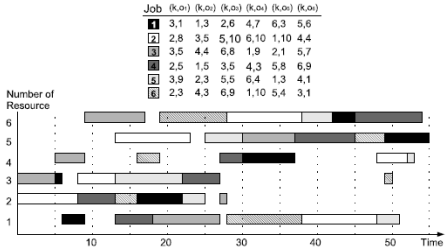
Roadmap

- Motivation
- Background
- Our Solution: Minerva
- Experimental Results
- Conclusion

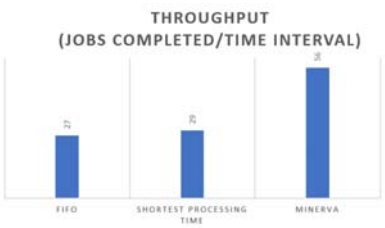
PURDUE UNIVERSITY

Throughput

(resource number, processing time)

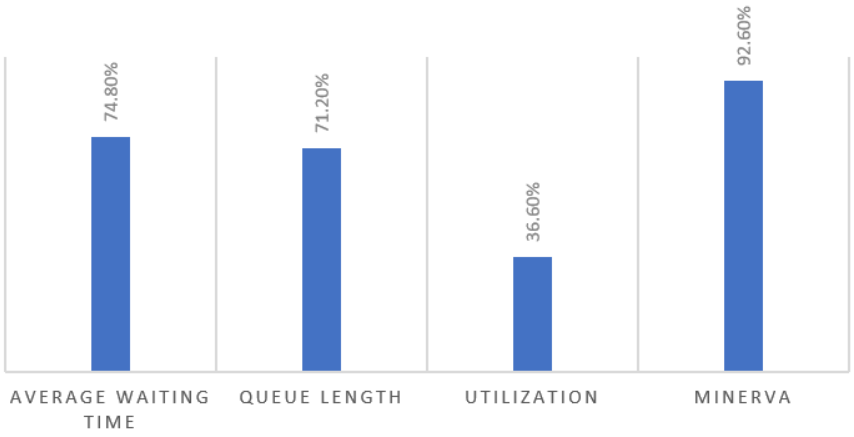


Widely used FT06 job shop scheduling benchmark



Results for the realistic extension of FT06 problem.

Bottleneck identification accuracy



Results for the realistic extension of FT06 problem.

Related work

- Other works on applying ML to solve JSSPs are limited
  - They address classical JSSPs only
  - Not applicable in the case of continuous arrival of jobs
  - Assume that the number of jobs is fixed
  - Most applicable to deterministic JSSPs only
  - None of them have addressed the problem of finding the bottleneck resource
- Different approaches to identifying bottlenecks are present in literature
  - Have limited accuracy
  - Require situation-dependent modeling

Conclusion

- Introduced Minerva, a novel technique to improve factory performance
  - Uses reinforcement learning to optimize scheduling
  - Uses neural networks to predict system bottlenecks
- Implemented Minerva on realistic extension of JSSP benchmarks and demonstrated the effectiveness
- Evaluated Minerva by comparing the results to prior techniques that are more rule-based and use single metric

Vision: Use of flexible data analytic techniques to improve factory operations under dynamically changing workloads