Learning to

Do things

Announcements

- We only have a class left
  - ↳ No readings for next week
  - → Talk about putting together all the things we talked about this semester
- Poster session
  - ↳ Expectations.

◦ Writeup: — Due 05/08

Missed a question/request in CampusWire.

- Briefly talk about fault tolerance / disaster recovery

- Q: How do large distributed systems get ∅ downtime?

Mostly they don't. Goal is many 9s of uptime but reality interferes: Even 99.9% is hard.

AWS service discounts [Region]

99 - 99.9%	10%
95 - 99%	30%
< 95	100%

Instance

99 - 99.5%	10%
95 - 99%	30%
< 95	100%

Q: What does 0 downtime mean?

↳ Depends. Generally goal is Availability

But

- want to avoid data loss but fine to not have immediate access
- Depends on system

Q: Disaster recovery?

Again, depends what you mean.

For network ops :- alternate in O(24) hours

Q: How?

↳ See last lecture.

Today: Overview on how to use data

- Over the last several weeks talked about several algorithms that
  - IDENTIFY SOME IMPORTANT CHARACTERISTIC
  - USE MEASURED VALUES/HISTORY/...

→ DQBarge/AutoPilot/Firm/C3/...

Control loops based on observations about performance/utilization/...

→ LDFI/SLFI

Failure injection based on observations about Job structure.

→ tprof/GMTA

Summarization based on likely anomalies

- In all cases: design based on some empirical or analytical observation about
  - Program structure & how failures impact them
  - Performance across dependencies
  - ...
- But how do we know that we picked the right factors/combined them in the correct way?

A: We don't.

At the end of the day, there is just too much possible data we can consider. All of it likely impacts performance/failure/...

Examples

→ Heat affects

↳ Performance (CPU throttling)

→ Likelihood of some types of failures

→ Silent data corruption

→ CPU batch/Identity affects performance/failures

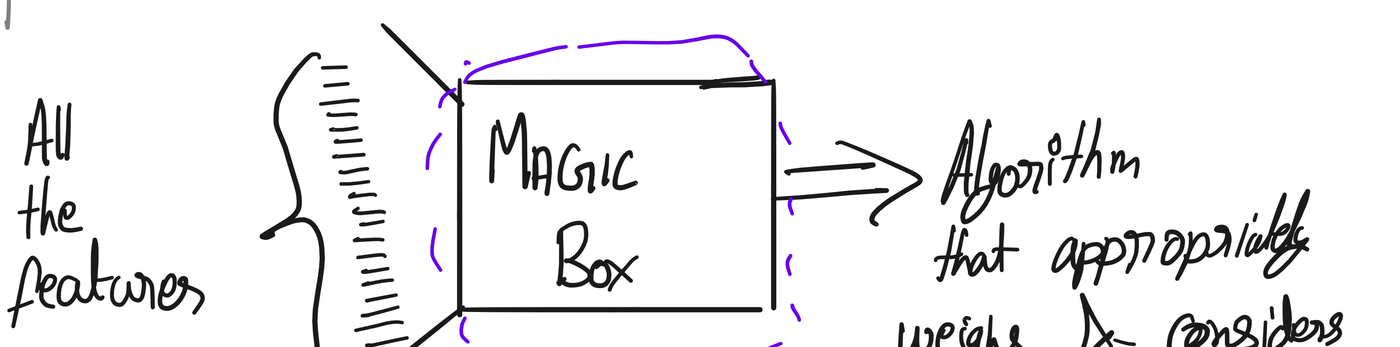
— Mercurial Cores (Hochschild et al)

→ Runtime used

→ ...

We just choose what seems important & design algorithms around it.

- The promise of ML for systems



them

TRAINING

Run into a few issues

- Learn from data. More features, more data
- Time & resources for training
- Cost of executing the resulting function
  - Often matters the most for frequently executed algorithms

Often solved by carefully engineering what features are provided & how they are represented

- A few steps forward from before
  - Consider more features
  - Don't have to assign importance
  - ...

- This keeps evolving. Used a few places in practice

- Power management at Google

- ...

- What does this have to do with traces

  ↳ Where else does the training data come from?

  → But for real

    ↳ Do we need to change them? How?

- Papers this class

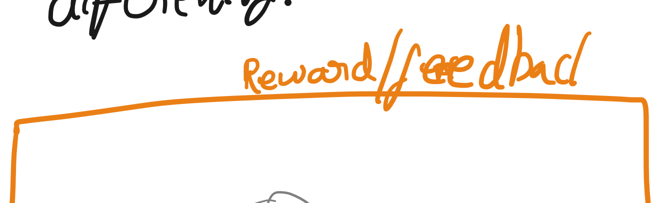
  ↳ Picked two I like
    Not necessarily the coolest/newest

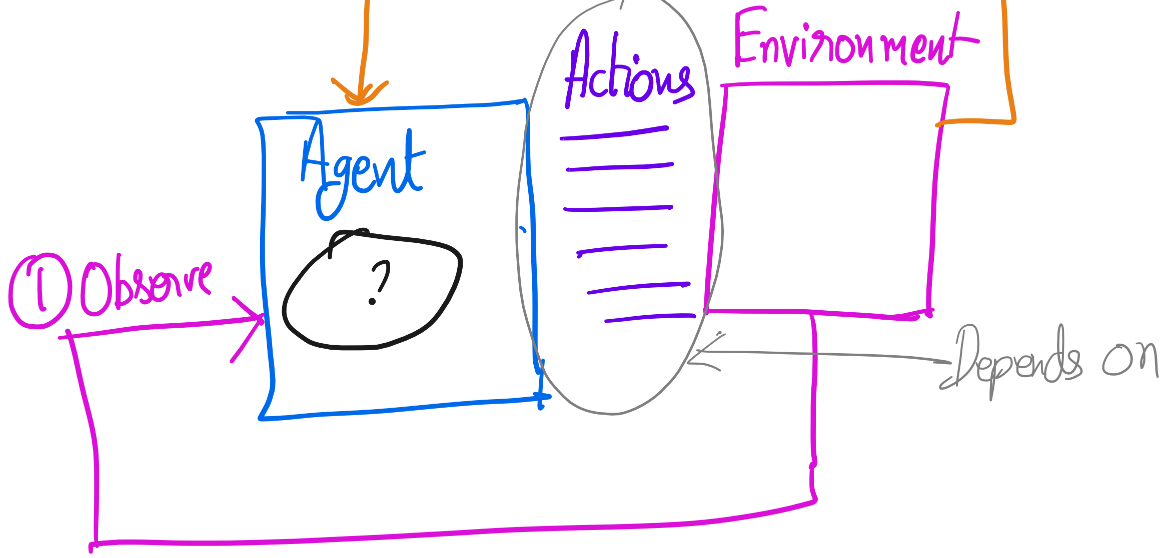
    [Someone, somewhere is looking at LLM for scheduling]

What both share: RL

Though used somewhat differently.

Roughly





How to model agent?

① Decima: Neural network
↳ GNN ← will come back to this

② Polyjuice: Lookup table

Trade-offs

	DECIMA	POLYJUICE
Time to Pick	Inference	Faster (HT)
Features Considered	T, J, C	⟨SP|D, A#⟩

Considered

Why is this the "correct" trade-off in this case?

Generalization/Effect on training data

PolyJuice

Actions

- When to read / wait
- What to read (committed/ uncommitted/...)
- When to make writes available
- Validate early?

↳ How much of txn is executed before checking.

Input to function

↳ Txn being executed

Does not consider

- Other transactions in the system

- Load etc.

Why?

- Let us look at assumptions when training

Utility : Maximize tput

Training assumption : Workload used for training ~ current workload

- # of transactions

- when they arrive
- ...

(+)

- Policy

⇒ What txns are running concurrently

But generalizability?

Decima

- Actions
  - Where to execute
    ↳ Think back to Distributed Resource Management on C3. Equivalent to what replica to pick
  - Parallelism
    ↳ How much work to have each

Coarser granularity than Polyjuice

executor do.

- Input:

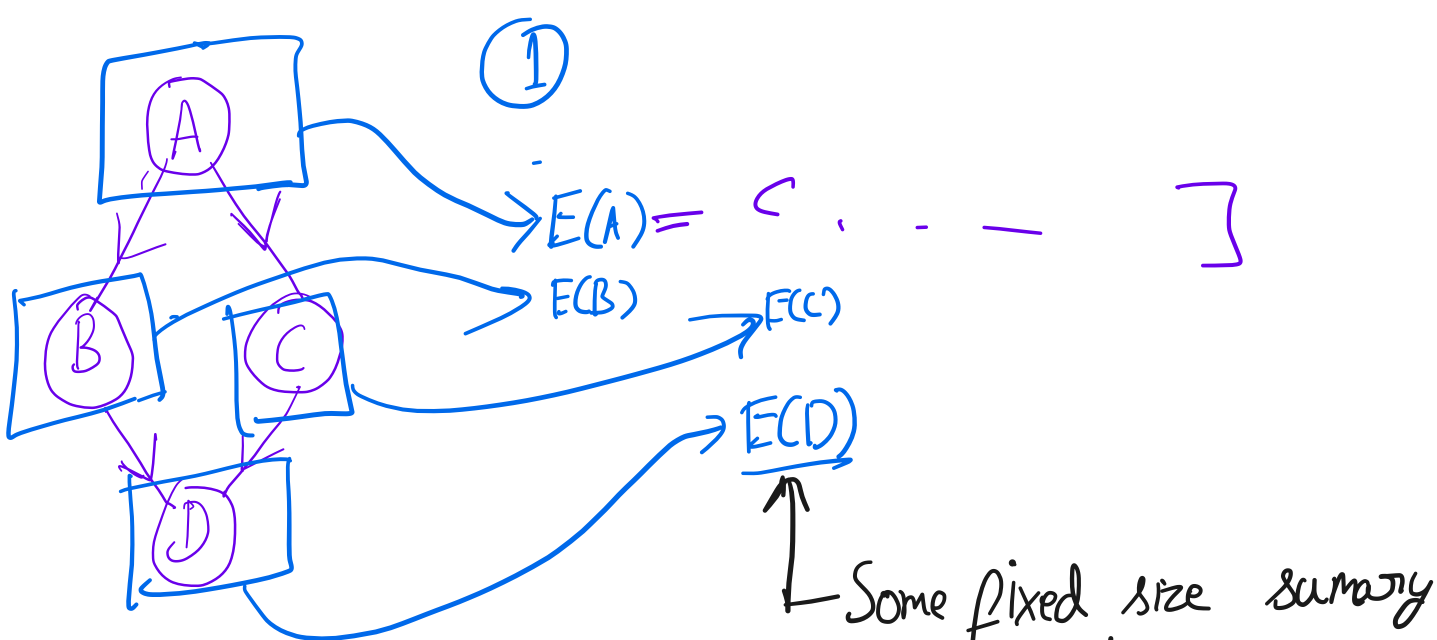
Task + Job + Other Jobs

Problem: How to represent this?

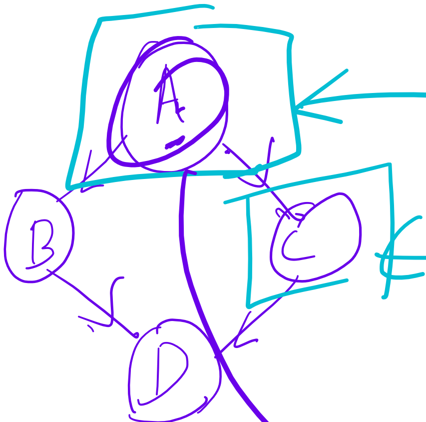
f : x, y, z → . . .

What if I have more than 3 inputs?

What if I have an unknown number of inputs?



of D. What does it contain?



But A's performance impacts C's performance which impacts D's..

$$E_J(A) = E(A) + G(E_J(B) + E_J(C))$$

$$E(B) + G(E_J(D))$$

Why?

How?

Bottom Line: $E_J(A) \dots E_J(D)$ contain some information about other downstream tasks.

$$G(\sum F(E_J(A))) \leftarrow Y_J$$

Says something about the job overall

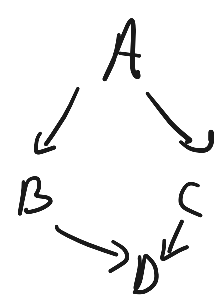
Same trick done across jobs ← Summary across cluster ← Z

Want to schedule A?

$sched(E_J(A), Y_J, Z)$

Assumptions we have made so far

- Go in reverse DAG order



- The need for G

-

Training assumptions:

- Jobs seen in training are "similar" to scheduled

??

Lessons/Promise of this approach.