SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2014

Using Reinforcement Learning to Optimize Storage Decisions

Ravi Khadiwala
Cleversafe

Topics

- What is Reinforcement Learning?
- Exploration vs. Exploitation
  - The Multi-armed Bandit
  - Optimizing read locations
- Solution Strategies
  - Thompson Sampling
- Practical applications and modifying problem constraints

Reinforcement Learning (RL)

- Consists of an **Agent** that interacts with an **Environment** and optimizes *overall* **Reward**
  - Agent collects information about the environment through interaction
- Standard applications include
  - A/B testing
  - Resource allocation

Uses in Data Storage

- Storage systems are dynamic environments
  - one size fit all parameters are difficult
- Situations that require adaptive intelligence:
  - Choosing an optimal set of nodes to read from
  - Scheduling housekeeping (potentially distributed) operations
  - Optimizing access to a shared resource in a decentralized manner

Formulating a RL problem

- Environment is usually based on an Markov Decision Problem (MDP) that isn't perfectly known by the Agent
  - *S* – Set of states
  - *A* – Set of actions
  - Rules for transitioning between states
  - Rules associating transitions with rewards
- One important addition: Rules describing what the agent observes

Formulating a RL problem

- Agent seeks to maximize reward
  - Chooses action from *A*
  - Observes from environment according to rule
  - Repeats

The Multi-armed Bandit

- By far the most studied problem in reinforcement learning
- N slot machines
  - Each with a unique, *fixed,* payout distribution
  - Each **round** you try one of the slot machines
  - Try to maximize your money over time!

Multi-armed Bandit formulation

- $B = \{B_1, B_2, \ldots B_n\}$, set of real distributions
- Equivalent to a one state MDP where
  - Action set: pulling levers 1..n
  - Reward: A draw from the associated distribution

SDC 14

Example: Reading with choices

- Simple scenario
  - Must read from 1 of of n nodes
  - Each node has a different latency distribution
  - Try to minimize user latency (without incurring wasted bandwidth)
- Mapping
  - Round: Each user read
  - Action: Choice of node
  - Reward: User latency

Evaluating Strategies

- A **strategy** is a scheme for picking actions/nodes
- Powerful evaluation criteria: **regret**
  - r^t is the reward observed at round t
  - r^* is the optimal reward
  - regret at round T is

$$T \cdot r^* - \sum_{t=0}^{T} r^t$$

- A good solution has average regret 0 as $T \to \infty$

Naïve Solutions

- Try all nodes once, then always pick fastest
  - May not converge to optimal node
- ε-greedy
  - Choose fastest with probability (1-ε) and choose randomly with probability ε
  - Will eventually learn optimal node
  - Average regret never goes to 0

Probability Matching

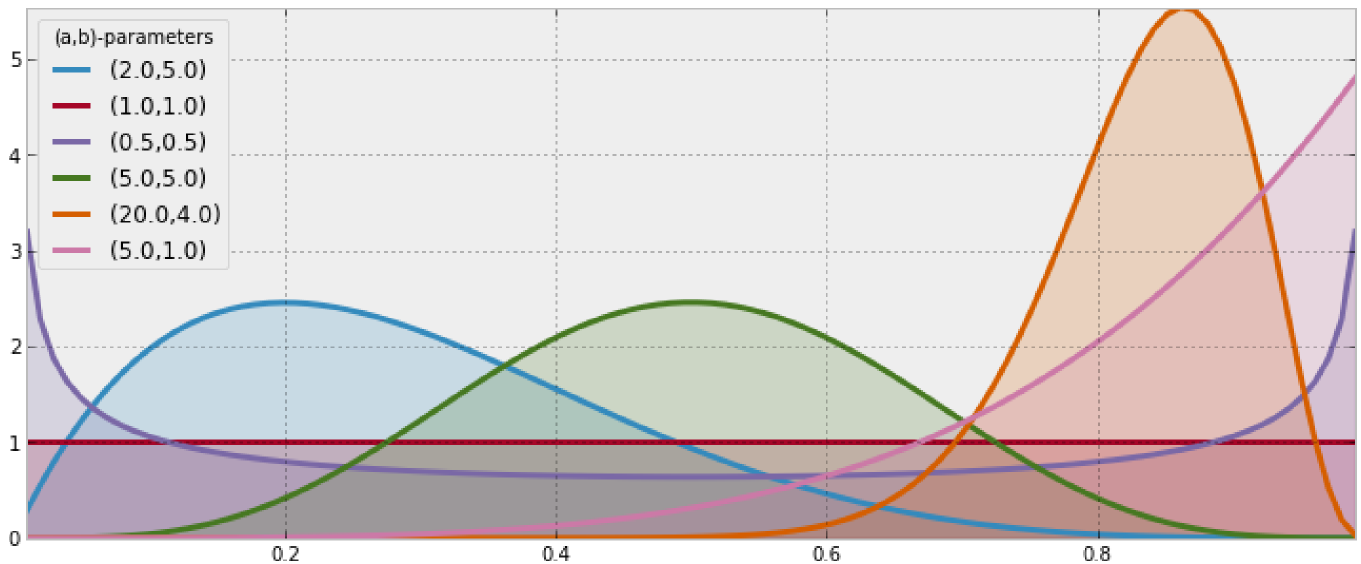
- Intuitively satisfying strategy
  - The further away an arm is from the current best, the less likely we are to explore it
- Thompson Sampling
  - AKA Bayesian Bandits
  - Approximately optimal
  - As confidence in arms increases, exploration decreases

Thompson Sampling

- Build a posterior distribution for each node based on observations
- Each round sample each distribution and choose the node that produced the 'best' sample
- Theoretically justified probability matching
- Advantages
  - Converges to optimal node
  - Easy to adapt to problem modifications

Beta Distribution

- The beta distribution has two parameters, α,β
  - α = number of successes
  - β = number of failures
  - α / (α + β) = expected value



Application 1: Read Ranking

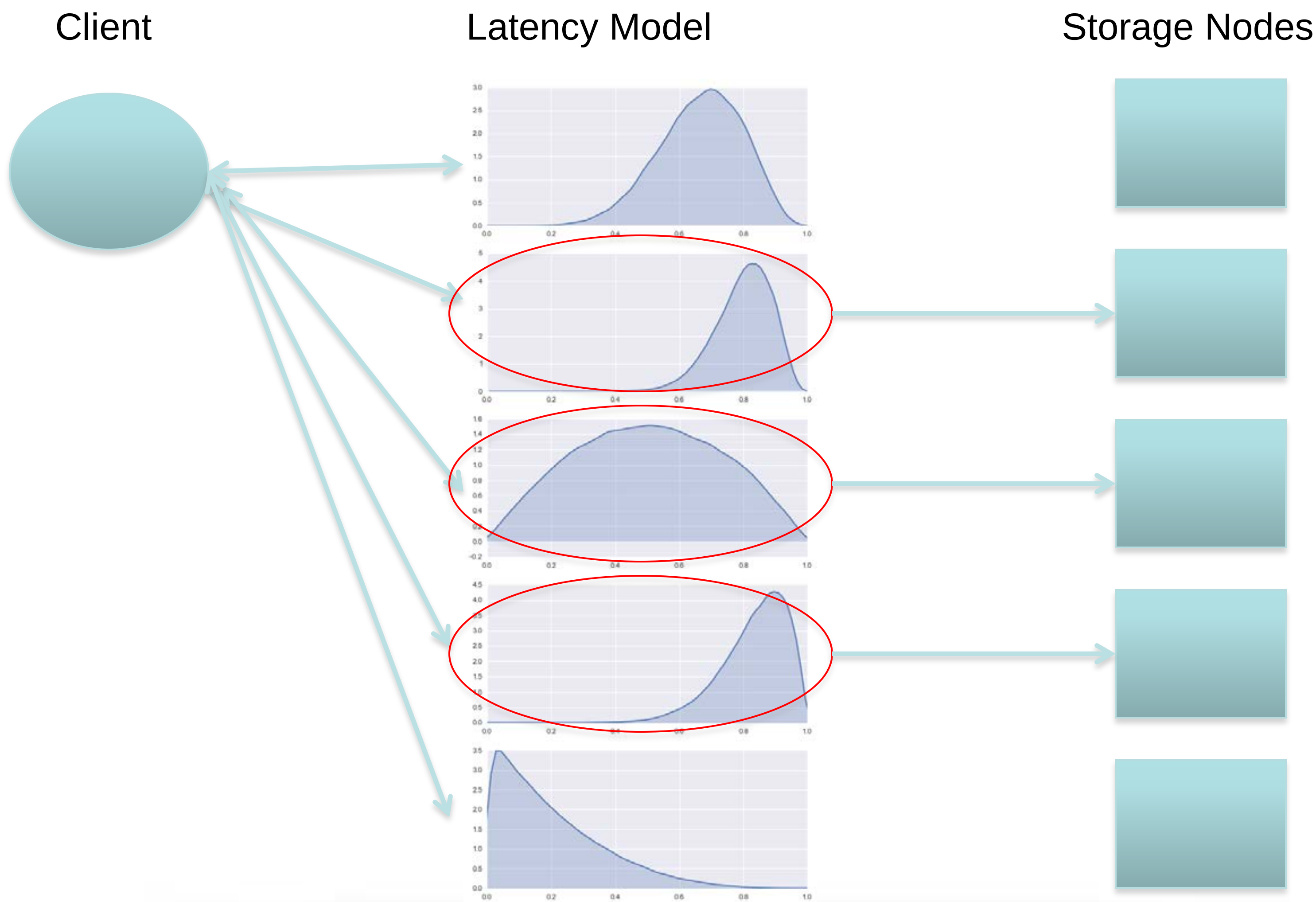
- Read from best k of n nodes
- Performance of nodes changes
  - Over time
  - Over access
- Optimize user latency
- Tradeoff with throughput

Constraint: Multiple selections per round

- Fairly easy to handle
  - Choose k best samples at each round
  - Consider a choice a ‘win’ if it is faster than the k^{th} slowest node historically
- Reduces to the one selection case

SDC 14

Application 1: Read Ranking



SDC 14

Constraint: Drifting rewards

- Relaxing the constraint of **fixed** distributions yields the **restless bandit** problem
  - Disks degrade
  - Network problems arise
  - Resource contention

Dynamic Thompson Sampling

- Simply limit $\alpha + \beta$
  - Limits total confidence in a distribution
- Simple to implement
- Practically very effective
  - Often out performs more complicated solutions

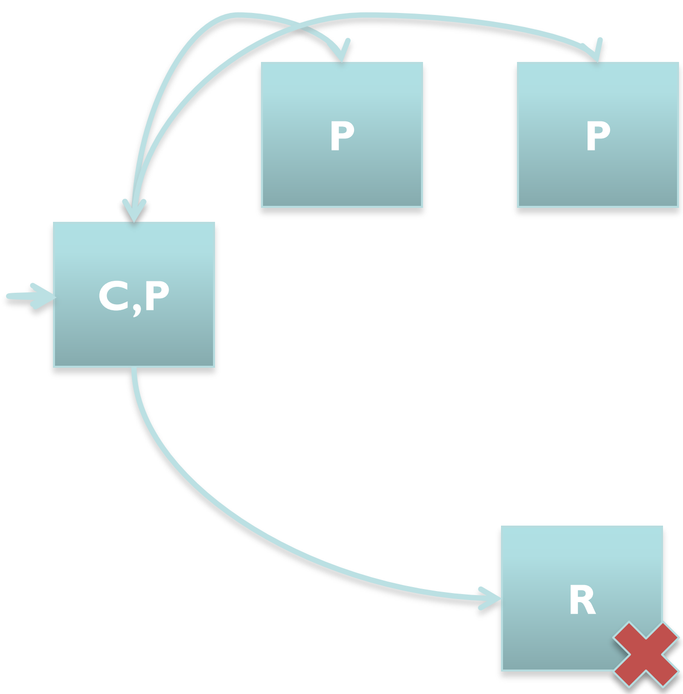
SDC 14

Constraint: Extra selections

- k selections not a tight requirement
- Make tradeoff between latency and throughput
- Compute predicted latency cost of a non-optimal selection
  - Send an extra read if it's worth paying the throughput cost

Application 2: Scheduling Housekeeping

- Rebuilding – repairing lost data
- Any node can discover that another node is missing data
  - **C**oordinator
  - **P**articipants
  - **R**ecipient



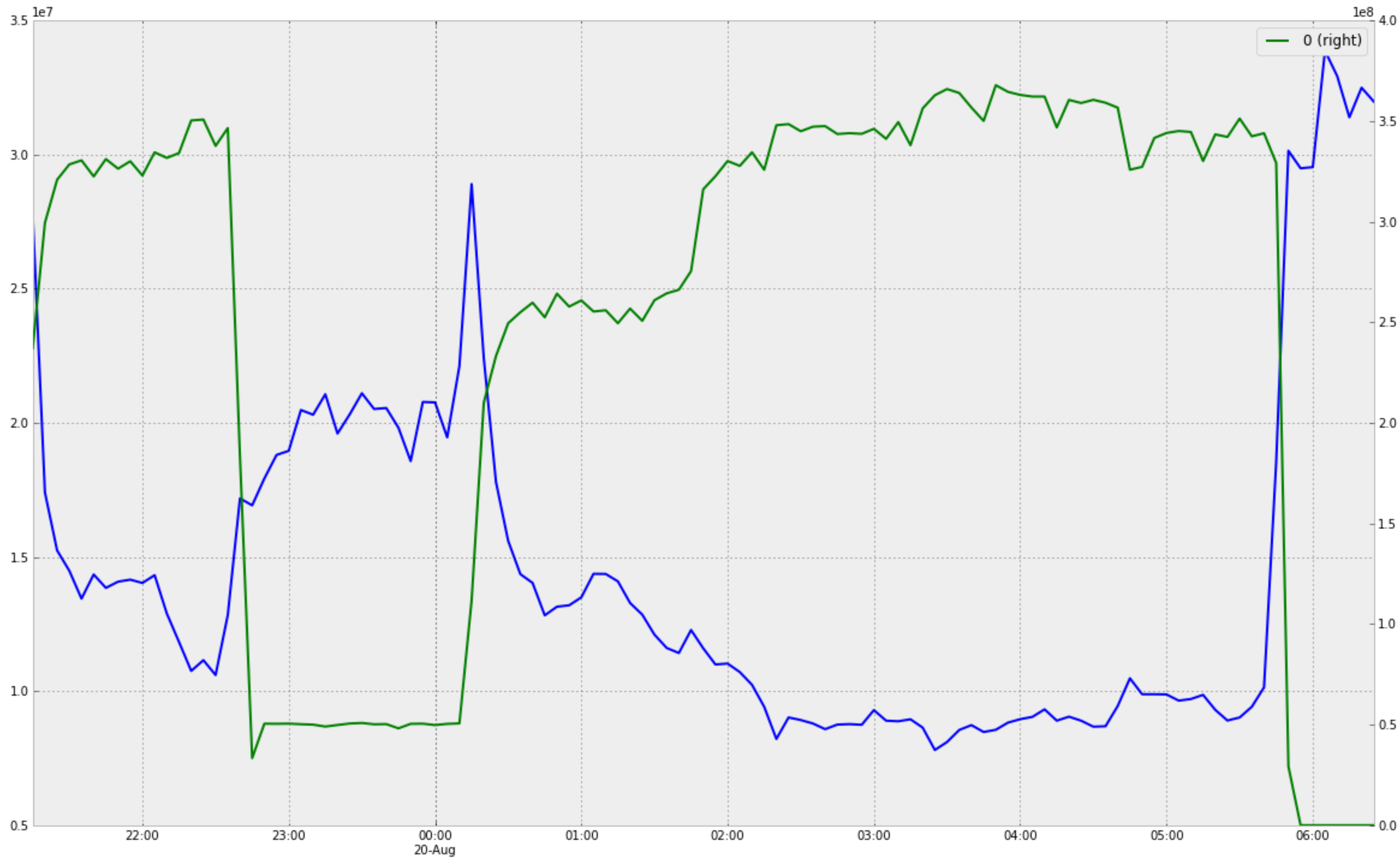
Rebuilding Architecture

- Expensive in an erasure coded storage system
  - Network bandwidth
  - CPU utilization
  - Disk utilization (reads + writes)
- Decentralized activity
  - Hard to know effect on global system

Formulating the scheduling problem

- Actions: A discrete set of rates the coordinator can rebuild at
- Rewards: A weighted combination of rebuild speed and observed throughput rate
  - Weight value of rebuild heavier the more unhealthy we perceive to be

Rebuild Rate v. Client I/O Rate



Aside: Parameter Tuning

- Hyper parameters?
- Tuning hyper parameters vs. parameters
  - Generality improves
  - Avoids overfitting
- Testing

Application 3: Decentralized resource management

- Many actors contend for a shared resource
  - Communication between actors may be infeasible
- May be limitations on resource's knowledge as well

Goore Game

- N voters, 1 Referee
  - Voters cannot communicate
- Each round players vote y/n
- Referee has a unimodal preference, f
  - There is some unique ratio of yes to no that maximizes f
- Goal: Voters converge on optimal ratio

Goore Game

- Thompson Sampling will eventually converge to ideal ratio
- Models many problems of imperfect information in distributed systems
  - Self organized performance optimized load balancing

Overview of RL

- Advantages
  - Planning for the general case is hard
  - Handle unforeseen scenarios gracefully
  - Good in systems with imperfect information
  - Relatively simple to implement
- Disadvantages
  - Can be unpredictable

Future Research Areas

- More complex models of bandit rewards
  - Contextual Bandits
- Generalizing to do RL on MDPs

Q & A