Codes for Storage with Queues for Access

Mentors: Dr. Emina Soljanin and Graduate Student Amir Behrouzi-Far

Dalton Burke[1] Elise Catania[2]

[1]University of Colorado Denver

[2]University of Rochester

DIMACS REU 2018

A romance of Coding Theory, Queueing Theory, Combinatorial Design, Python, Graph Theory, Linear Programming, and more.

Table of Contents

Codes for Storage with Queues for Access

1. Introduction
   - Research Problem
   - Other Policies
2. Combinatorial Designs
   - The Fano Plane
   - Sequence Example
   - BIBD Sequences
   - Optimal k Value
   - A More Realistic Model
3. Service Capacity Problem
   - Introduction
   - Potential Approaches
   - A Result
   - Next Up

Businesses depend on computers to process large data sets. Valuing time, we seek an efficient process for completing jobs. We want to squeeze as many jobs out of the system as fast as possible.

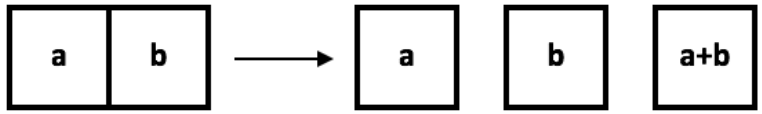
Lots of jobs. Lots of queues. Lots of servers.
Lots of plots.

We parallelize, divide the work between different servers.

- Issue of stragglers

Coding makes the system robust to stragglers.

- Split the job into k pieces and encode them into n tasks. (note: $k \leq n$)
- Send each of the n tasks to a server.
- k out of n tasks completes the job.

Ex: 

Assumptions:

- Each task is the same size.
- All of the computers have the same resources.

So how do we choose which n servers receive the tasks?

Other Policies

There are many studied ways to set up a distribution policy:

- JSQ: Join the n shortest queues.
- Power-of-d: Pick d queues randomly, choose the shortest n of those.
- Round Robin: Cycle through all servers in order, n at a time.
- Random: Choose the n servers randomly.

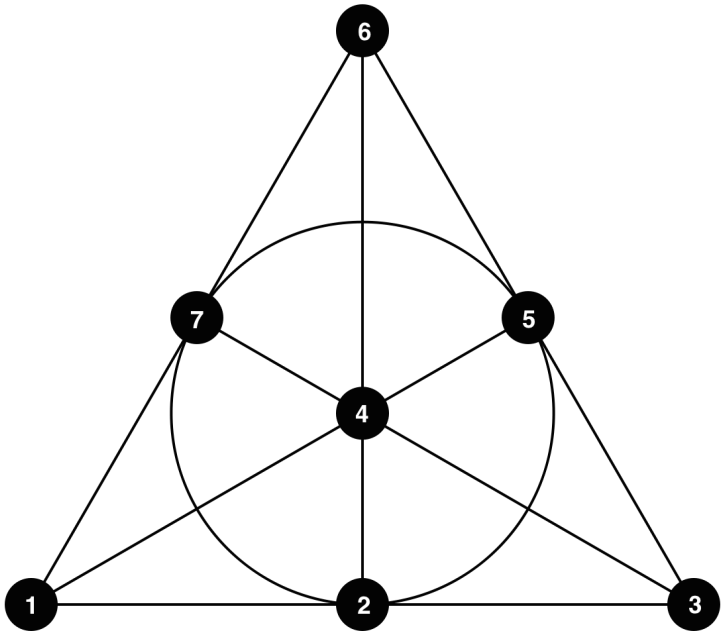
- Fork-rejoin: Set n equal to the number of servers (hard to compare).

Combinatorial Designs

(7,3,1)-BIBD

The Fano plane represents a (7,3,1)-balanced and incomplete block design.

- It consists of 7 lines and 7 points such that there are three points on each line.
- Using the Fano plane as a model, we create a situation where tasks are sent to 3 out of the 7 total servers at a time.



Combinatorial Designs

(7,3,1)-BIBD

From the Fano Plane, we form the set
$\{123, 145, 167, 246, 257, 347, 356\}$.

For convenience, we map
$[123, 145, 167, 246, 257, 347, 356]$ onto $[0,1,2,3,4,5,6]$.

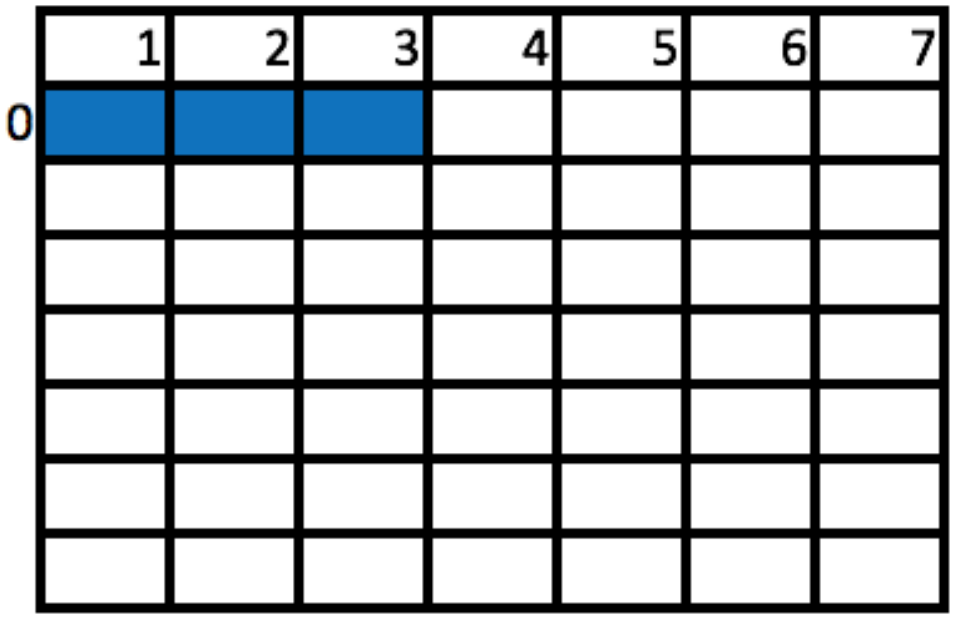
Are certain ordered sequences more efficient than others?

Combinatorial Design

Sequence Example

We will show the sequence $\{0,1,2,3,4,5,6\}$.

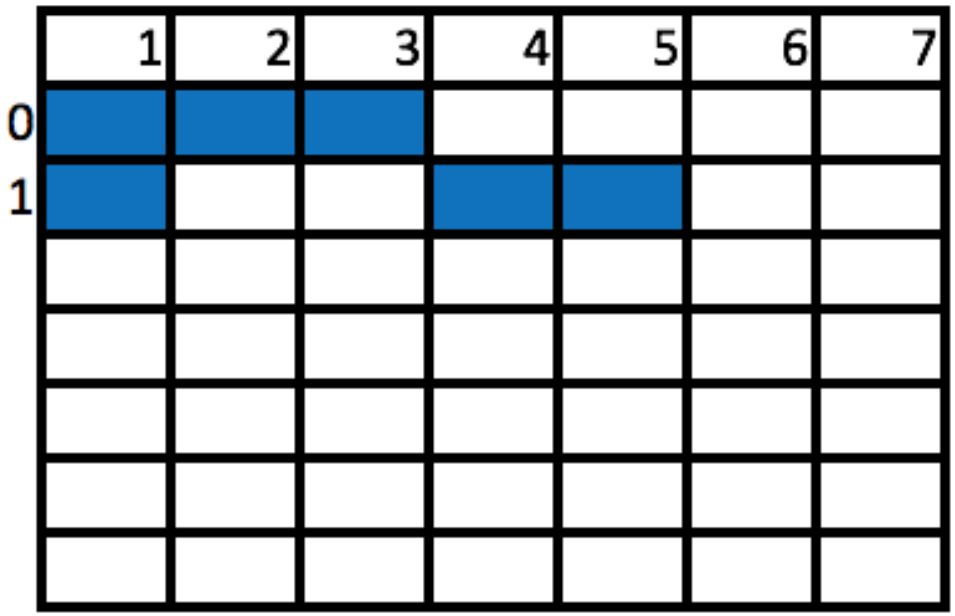
- Remember $\{123, 145, 167, 246, 257, 347, 356\}$ maps to $\{0,1,2,3,4,5,6\}$.



Combinatorial Design

Sequence Example

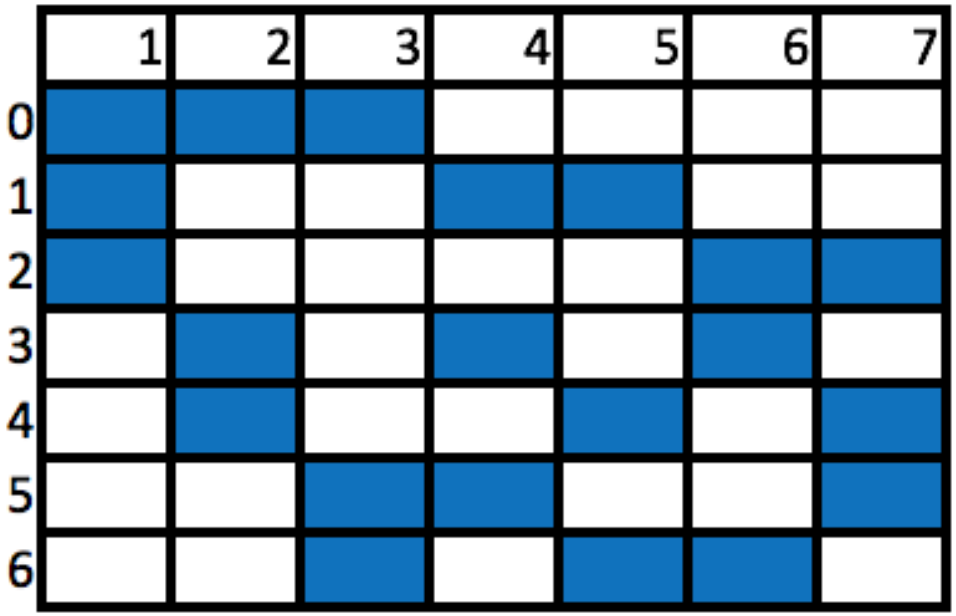
- Remember $\{123, 145, 167, 246, 257, 347, 356\}$ maps to $\{0,1,2,3,4,5,6\}$.



Combinatorial Design

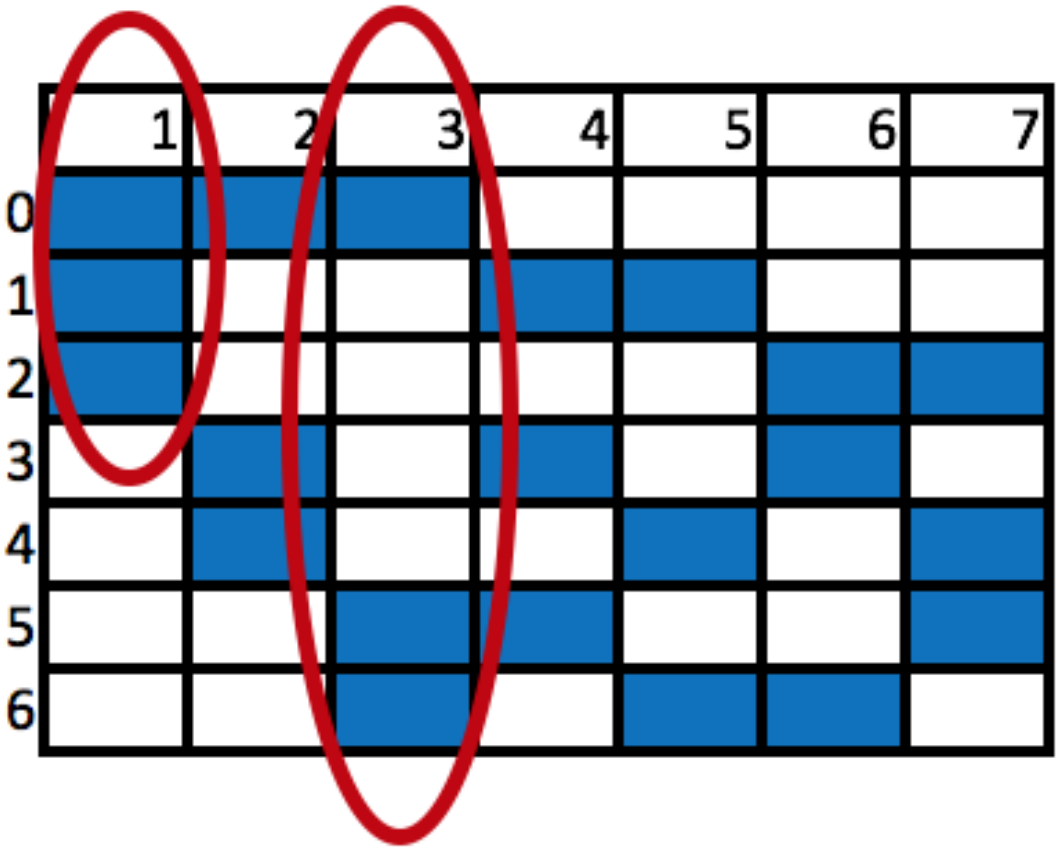
Sequence Example

- Remember $\{123, 145, 167, 246, 257, 347, 356\}$ maps to $\{0,1,2,3,4,5,6\}$.



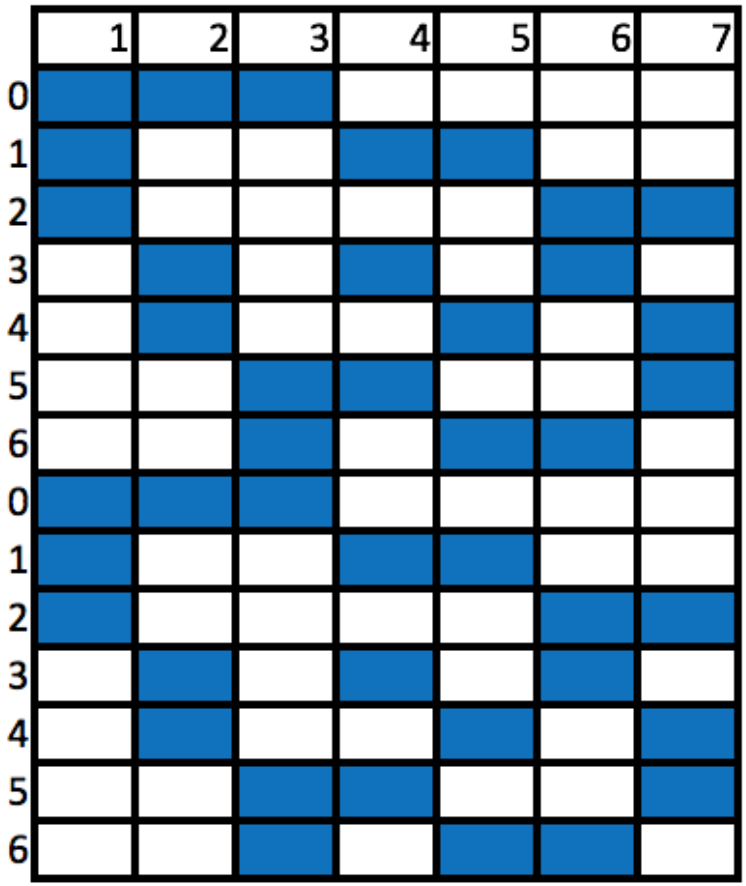
Combinatorial Design

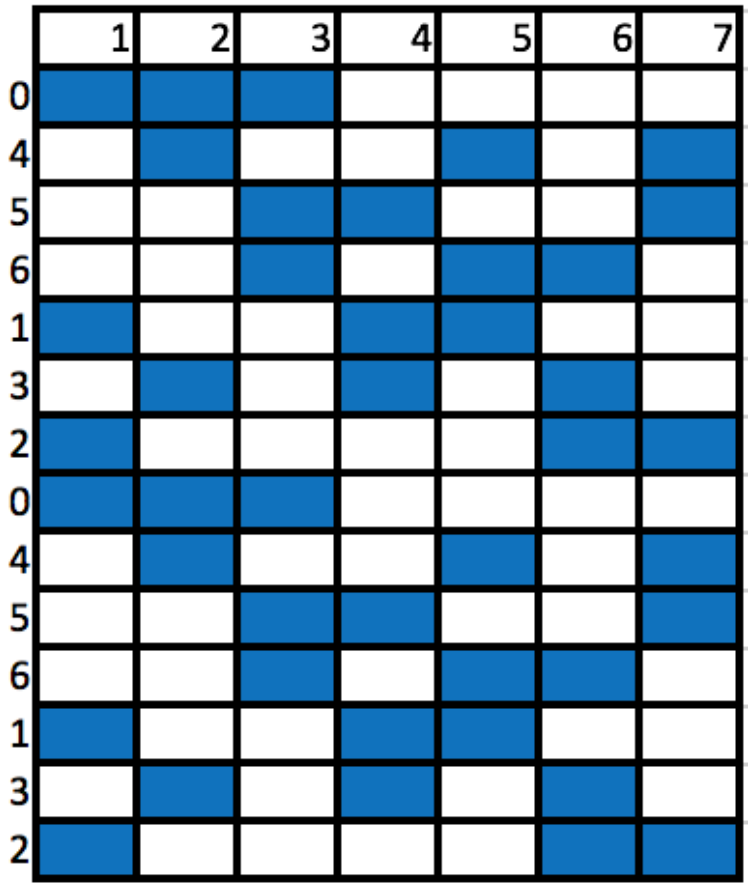
Sequence Example



Combinatorial Design

Sequence 0,1,2,3,4,5,6 Vs. Sequence 0,4,5,6,1,3,2



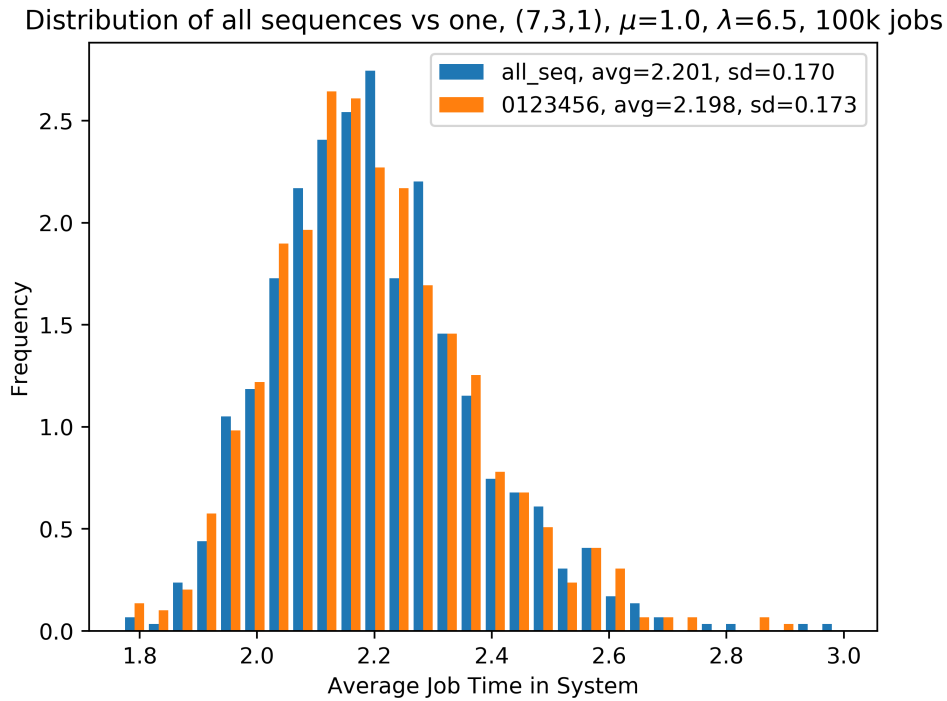


Does the Sequence Matter for BIBD?

Using the Simulation

We ran all 720 unique sequences and compared it to the sequence $\{0,1,2,3,4,5,6\}$ run 720 times for 100,000 jobs, k=1, μ=1.0, and λ=6.5.

Note: μ and λ denote service rate and arrival rate, respectively.



Which k Value is Optimal?

(7 3 1)-BIBD

Consider the different cases: $k = 1, 2, 3$. Why would one case be more advantageous than the other two?

(Remember: We split the job into k tasks and encode into n tasks)

$\underline{k = 1}$: Can get around stragglers, but one server must process the whole job.

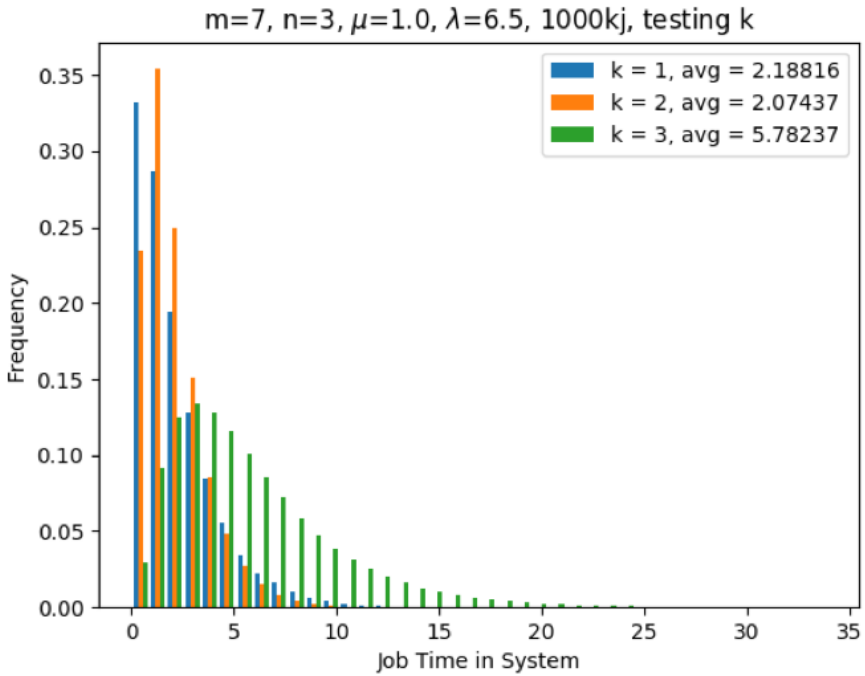
$\underline{k = 2}$: The job is split into two tasks and we wait for two servers to finish their tasks.

$\underline{k = 3}$: The job is split into three tasks and we wait for three servers to finish their tasks.

Which k Value is Optimal?

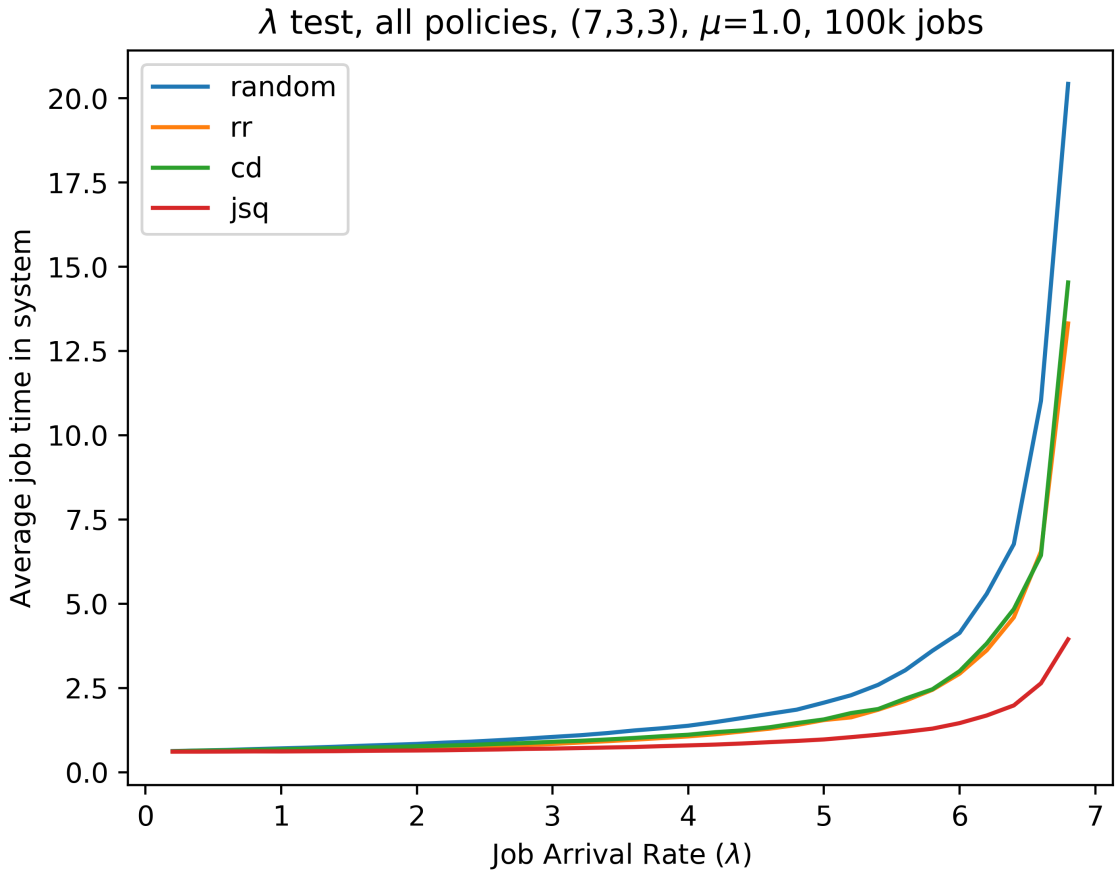
(7 3 1)-BIBD

We ran one million jobs for the (7 3 1) BIBD for k=1,2,3.



- We see that for the (7 3 1) design, k=2 is optimal.
- From this information, we suspected that a middle k value would be most efficient for other BIBD schemes as well.

Round Robin Vs. Combinatorial Design



Combinatorial design and round robin perform about the same (round robin is slightly better).

A More Realistic Model

In reality, not all jobs will be unit size. To account for this, we split the service time into two terms, depending on:

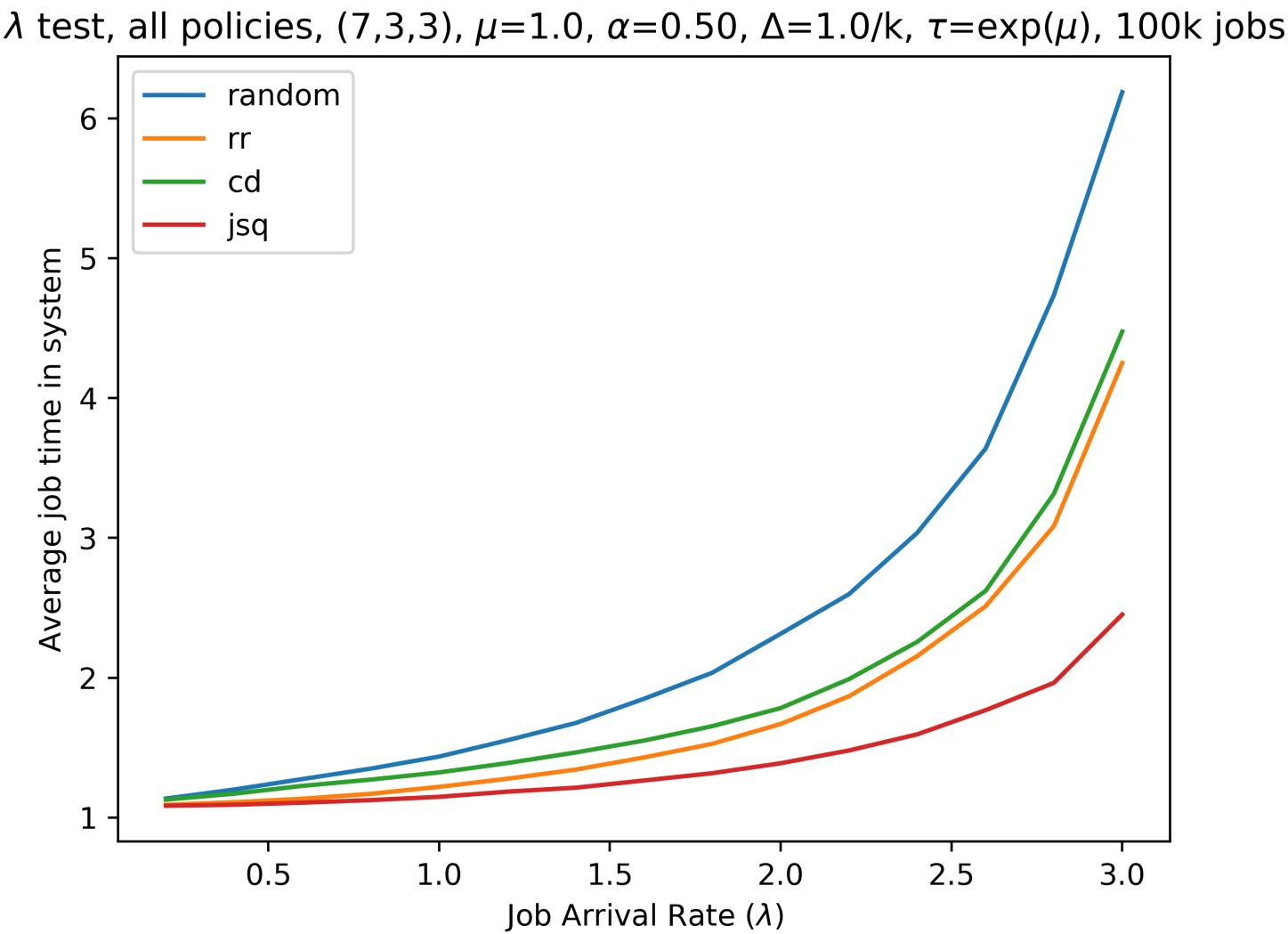
- The "size" of job i, Δ_i
  - Constant
  - Random (exponential)
  - Random (pareto)
  - Mice and Elephants
- The "delay" caused by server j, τ_{ij}
  - Random (exponential, μ)
- A balancing coefficient, $0 \leq \alpha \leq 1$

We use these together for a new service time: $\alpha * \frac{\Delta_i}{k} + (1 - \alpha) * \tau_{ij}$

How do the policies stack up here?

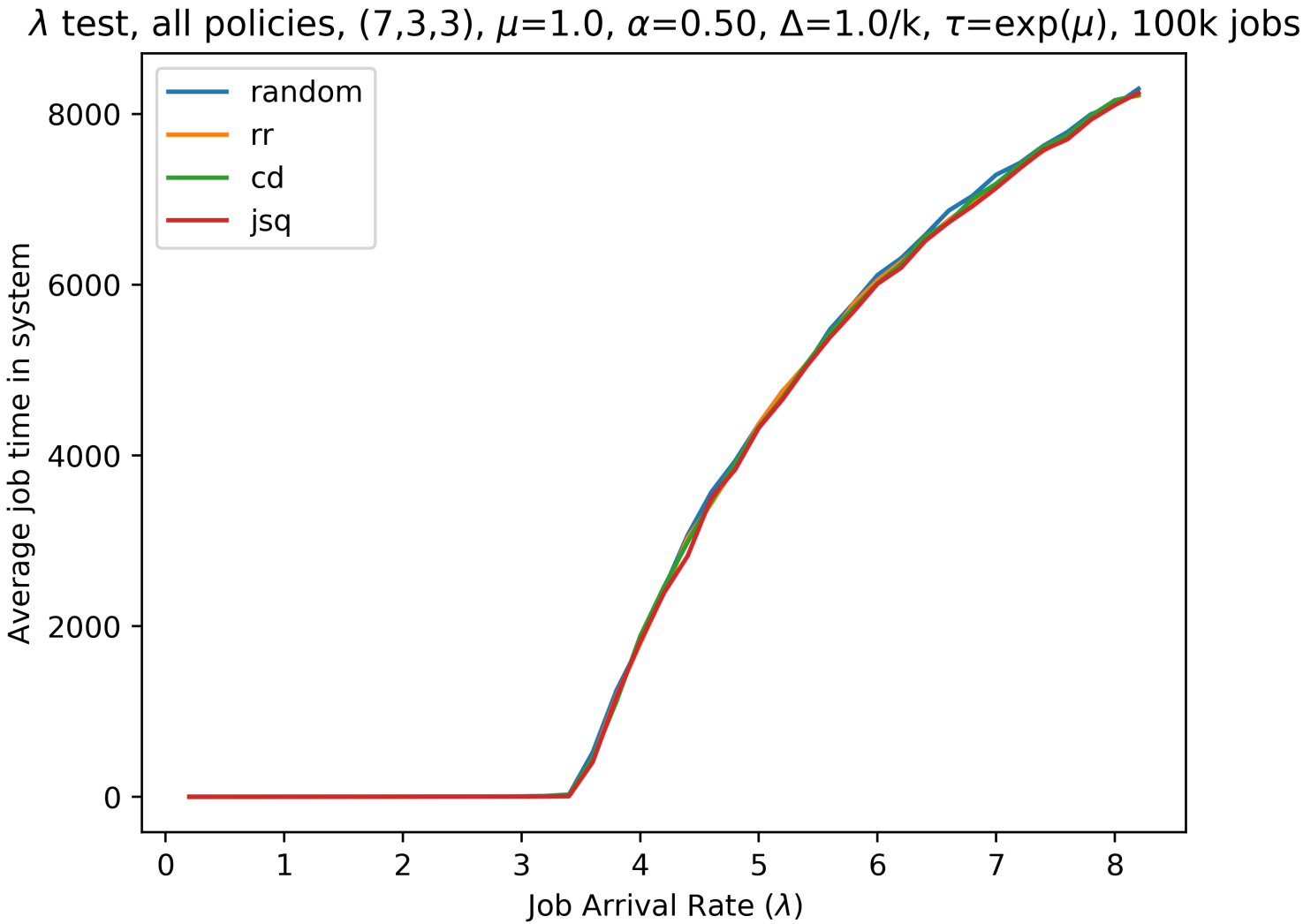
Constant Δ

$k = 3$



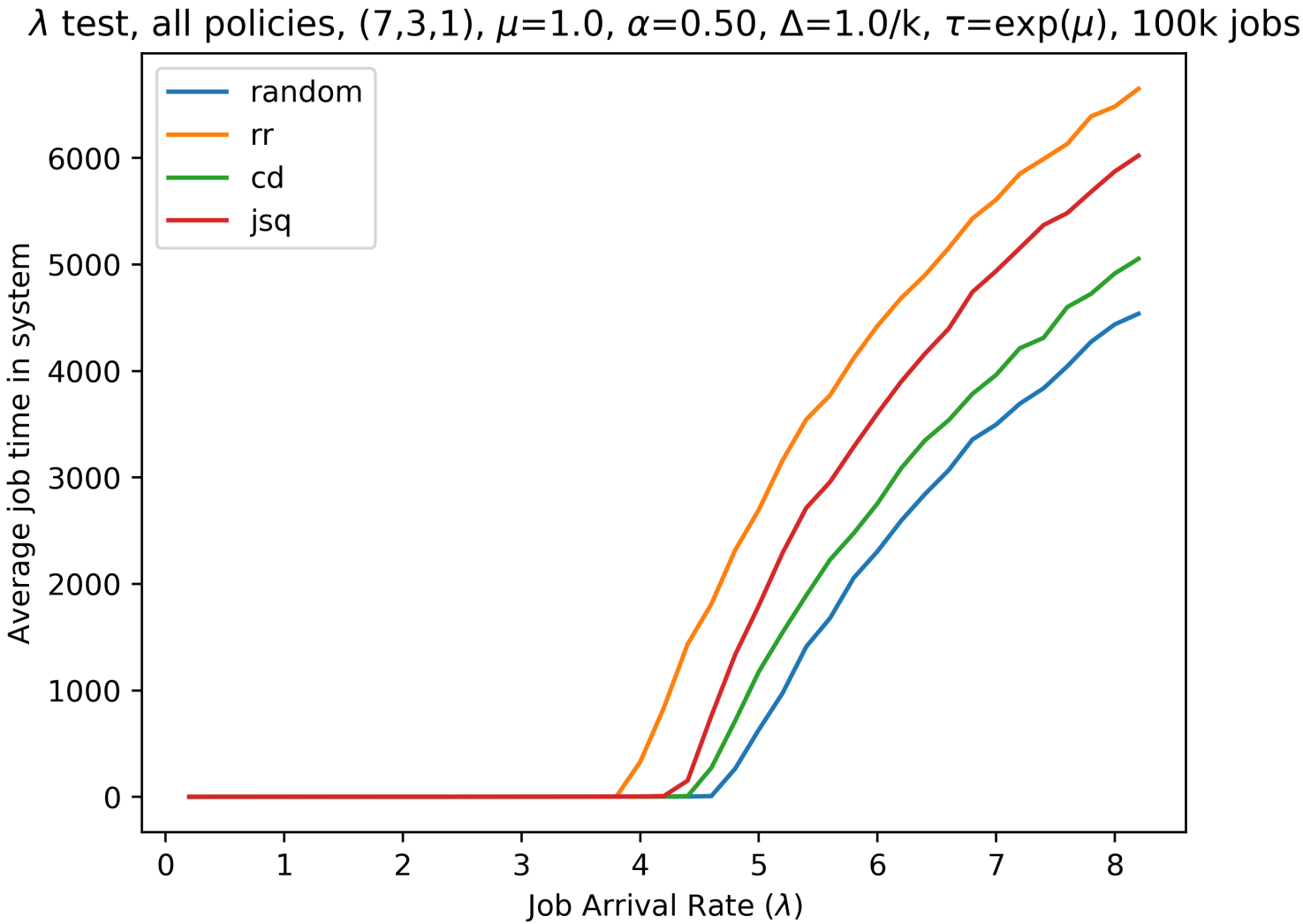
Constant Δ

$k = 3$



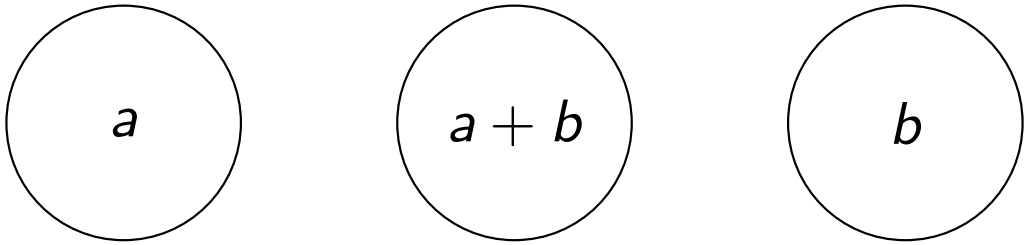
Constant Δ

$k = 1$?????



Service Capacity Problem

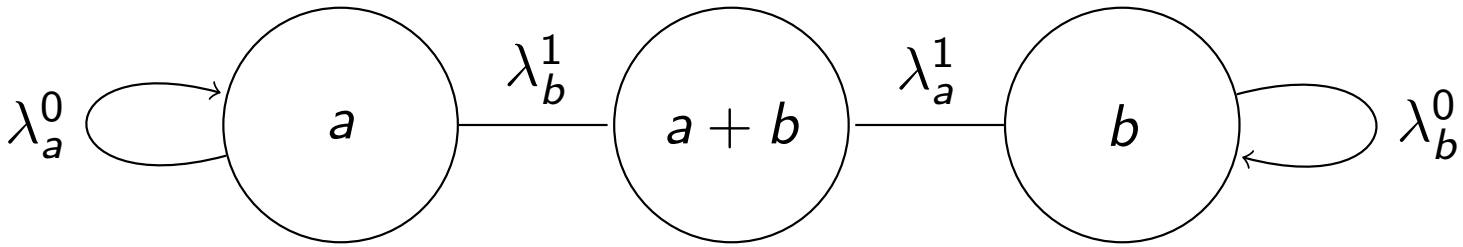
Suppose we have two *files*, a and b. We might have a server for each (with service rate $\mu = 1$), and a third containing a coded mash of both.



What kind of demands for a and b (λ_a and λ_b respectively) can this system handle?

Service Capacity Problem

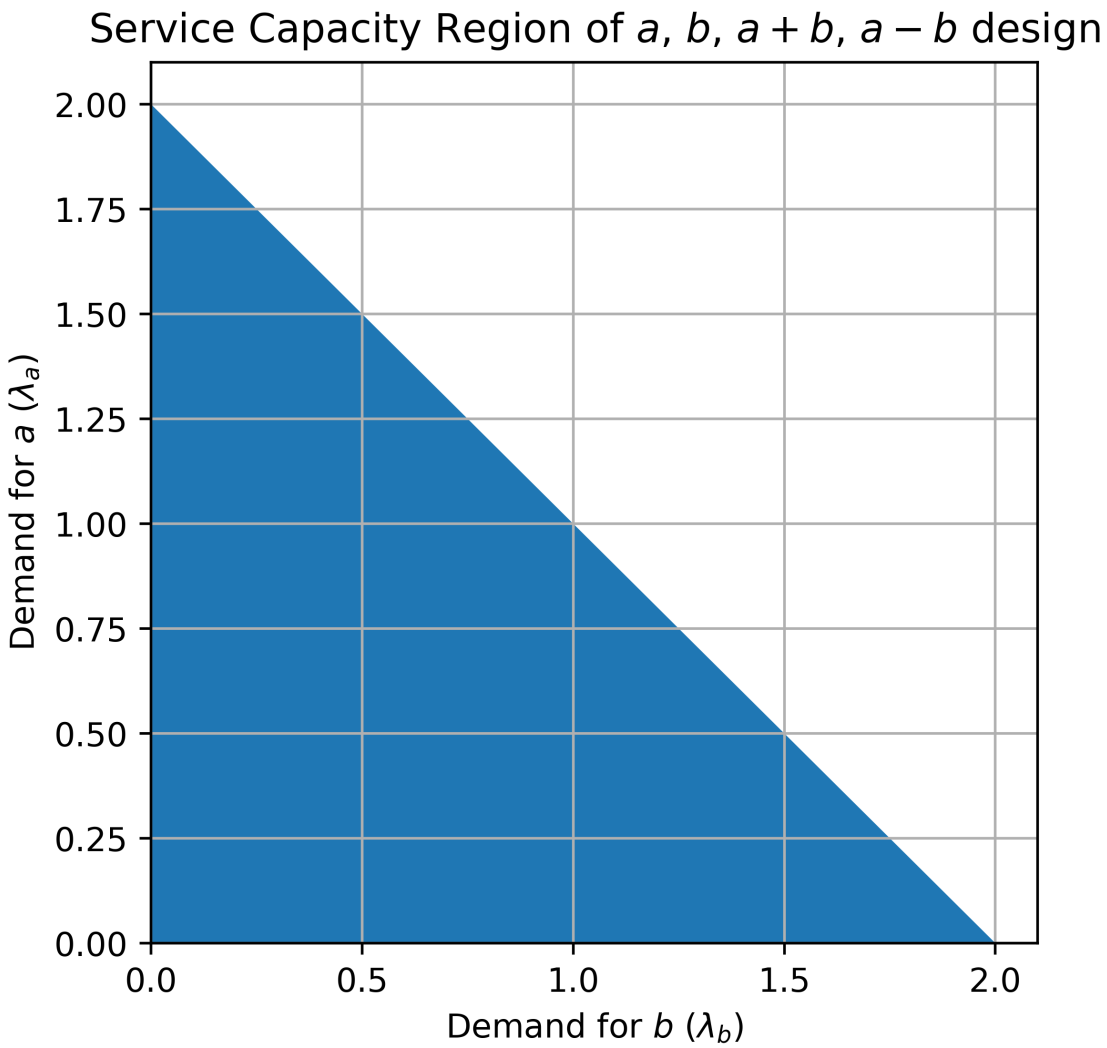
A graph representation of such a system can be handy.



- Vertices for servers, where a single file may be held (systematic), or a coded mash of files (coded)
- Edges can be used to denote when a single file may be extracted from incident vertices.

Service Capacity Problem

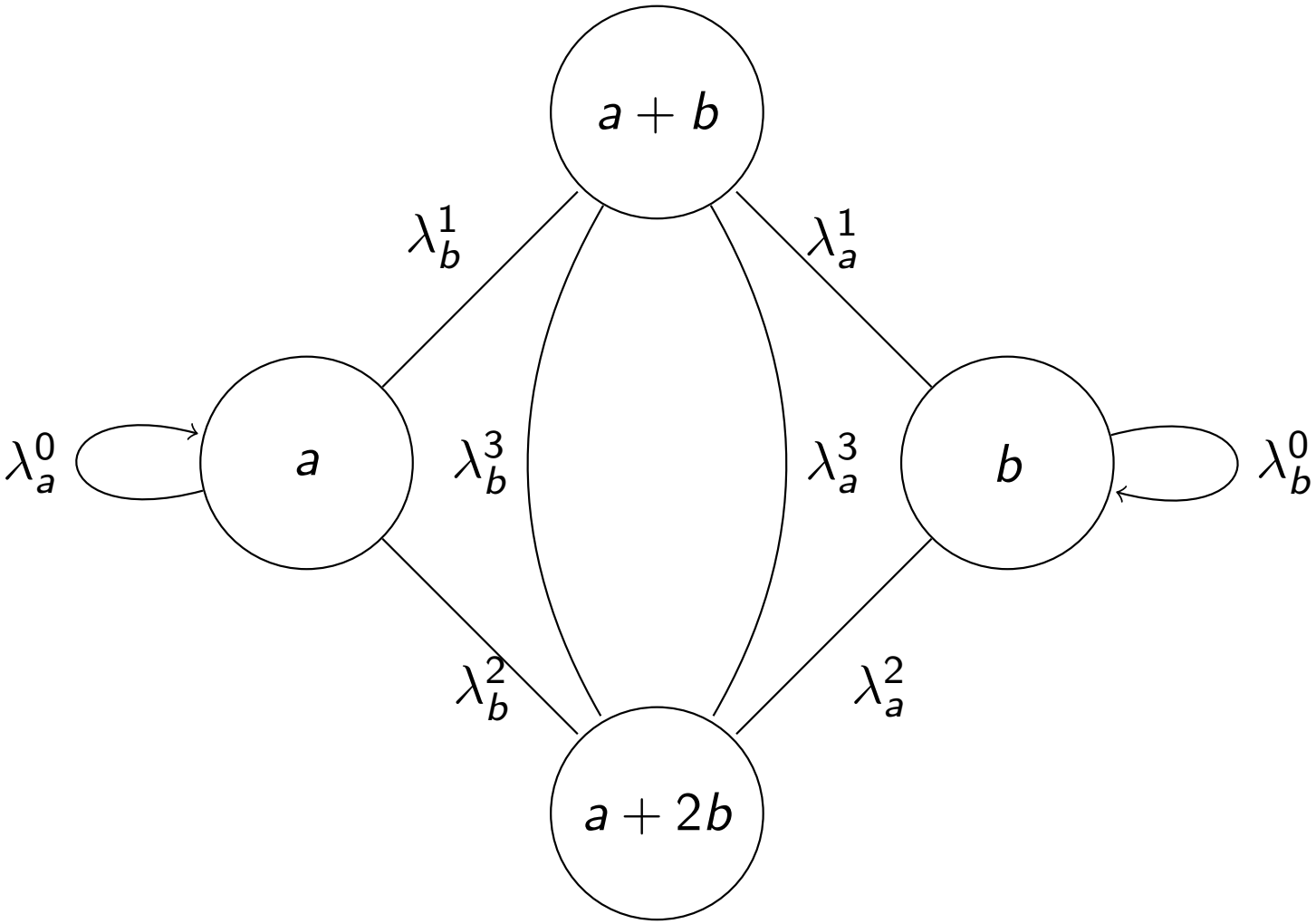
More plots



$$\lambda_a + \lambda_b \leq 2$$

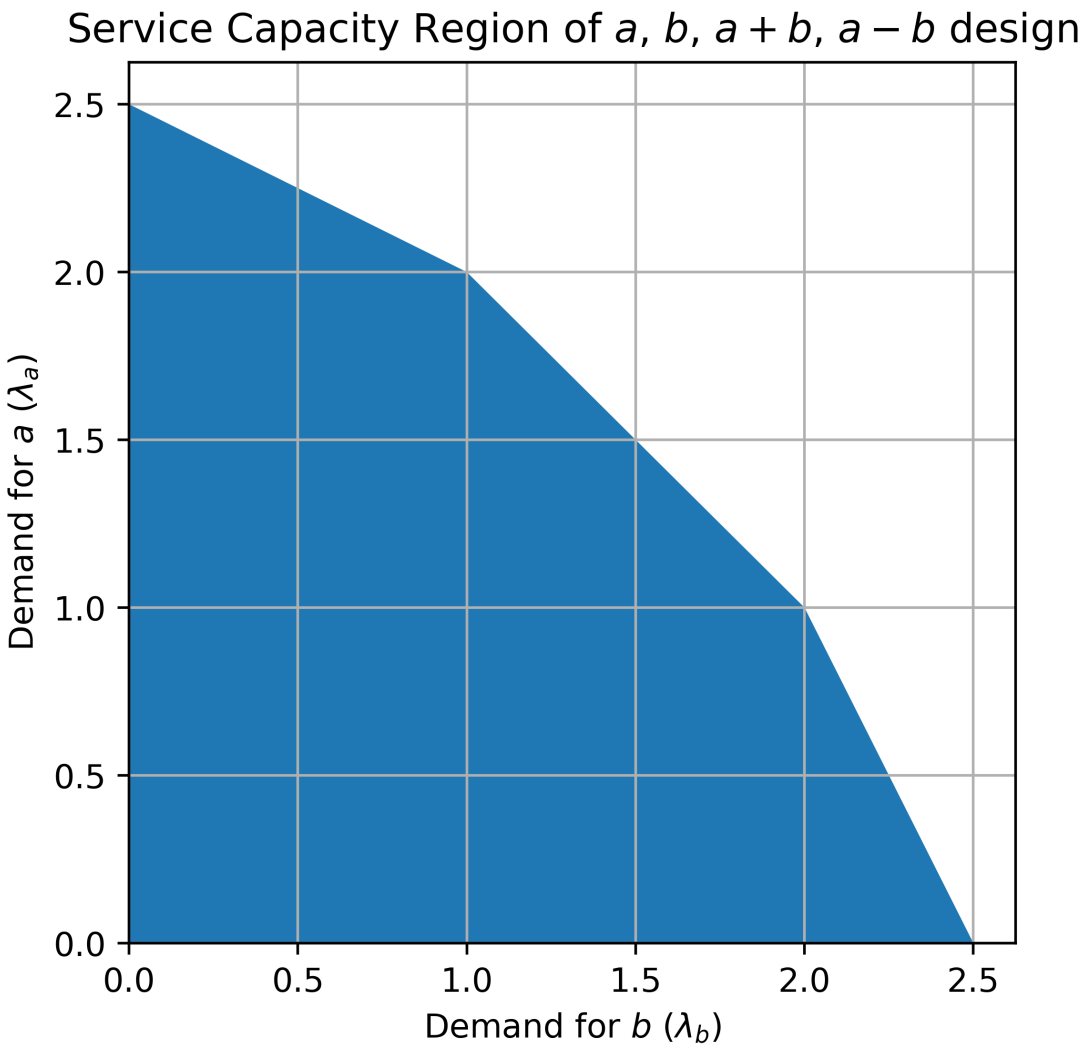
Service Capacity Problem

Another example



Service Capacity Problem

More plots



$$\lambda_a \geq 1 \text{ and } \lambda_b \leq 1$$
$$2\lambda_a + \lambda_b \leq 5$$

$$\lambda_a \geq 1 \text{ and } \lambda_b \geq 1$$
$$\lambda_a + \lambda_b \leq 3$$

$$\lambda_a \leq 1 \text{ and } \lambda_b \geq 1$$
$$\lambda_a + 2\lambda_b \leq 5$$

Service Capacity Problem

Potential Approaches

How can we derive these bounds for K files and N servers?

- Vertex Covering
- Network Flows
- Linear Programming
- General Analysis of System

Achievability vs. True Capacity

Service Capacity Problem

Moving forward

We know that

- To get one unit out of a systematic server, only one server is needed.
- To get one unit out of a parity server, many are required.

Using systematic nodes first gives us better bounds, this combined with some efficient load balancing leads to a result.

Service Capacity Problem

A Result

In general we would be working with N servers, and K files. Each file has one systematic server dedicated to it, the remaining $N - K$ are coded. If there are more coded servers than systematic, we showed that

Given $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_K$, and $\lambda_j \geq \mu \geq \lambda_{j+1}$:

$$\sum_{i=1}^{j} \lambda_i + \sum_{i=j+1}^{K} \frac{\lambda_i}{(i-1) * i} \leq \mu * \left(j + \frac{1}{j} + \frac{N-K-1}{K} \right)$$

"Nice." - Neekon Vafa

Service Capacity Problem

Next Up

What do we do if we have fewer coded servers than systematic?
In the last case, the coded nodes were utilized when demand for a certain file was more than its systematic server could handle.
We need to combine parts of K coded servers to obtain a file, but, if we have less than K coded servers, we need a more complex extraction scheme.

Coming Soon.

Acknowledgements

- Thank you to the NSF for supporting our work through grant CCF-1559855 and CCF-1717314.
- Thank you to our mentors Dr. Emina Soljanin and graduate student Amir Behrouzi-Far for guidance and support throughout our project.
- Thank you to DIMACS for providing the opportunity for our research.

References

- Aktas, Mehmet, et al. On the Service Capacity Region of Accessing Erasure Coded Content. 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2017, doi:10.1109/allerton.2017.8262713
- Stinson, Douglas Robert. *Combinatorial Designs: Constructions and Analysis.* Springer, 2004