

CS 262A: Advanced Topics in Computer Systems

Ion Stoica

January 22, 2025

Course Information

Course website:

- <https://ucbsky.github.io/ucbsky-cs262a-spring2025/>
- It is on github so you can contribute content!

Schedule and reading list are subject to change

Who are we?

Ion Stoica

- Professor, UC Berkeley (AI Systems, Cloud Computing)
- Co-founder: Databricks, Anyscale, Conviva
- OSS involvement: Apache Spark, Ray, Apache Mesos, Ray, vLLM, ...



Who are we?

Audrey Cheng

- TA
- PhD Student (Databases)
- Research focused on transaction scheduling, performance isolation



What is System Research about?

Manage resources:

- Memory, CPU, storage
- Data (database systems)

Provide abstractions to applications:

- File systems
- Processes, threads
- VM, containers
- Naming system
- ...

This Class

Learn about systems research by

- Reading several seminal papers
- Doing it: work on an exciting project

Hopefully start next generation of impactful systems

- E.g., Apache Mesos was a class project in such a class

Appreciate what is Good Research

Problem selection

Solution & research methodology

Presentation

What do you need to do?

Research oriented class project

- Groups of 2-4 students

Paper reading

Paper presentations

Research Project

Investigate new ideas and solutions in a class research project

- Define the problem
- Execute the research
- Write up and present your research

Ideally, best projects will become conference papers (e.g., OSDI/SOSP, NSDI, EuroSys, MLSys, ...)

Research Project: Steps

We'll distribute a list of projects

- You can either choose one or come up with your own

Pick your partner(s) and submit a one-page proposal describing:

- The problem you are solving
- Your plan of attack with milestones and dates
- Any special resources you may need

Poster session

Submit project report

Paper Reading

All students are required to read all papers

All students are required to submit “reviews” for each paper:

- Answer short questions on google form (see next slide)
- We will send out how to signup for papers later today

Paper Reading: Key Questions

- What is the problem being solved?
- How was this problem solved previously?
- What is the main idea?
- What are the key results?
- What are the main limitations of this paper?
- Why did this paper have an impact?
(or, Do you think the paper will have impact?)

Submissions: Will send out a google form for every paper that you need to fill in (will close it 10min before the class)

Distributed Shared Memory

Countless papers in 1990s:

- Very compelling abstraction
- Many hard challenges, so many researchers worked on it

Today

- Few systems using distributed shared memory, if any
 - Note: Comeback in the context of disaggregated memory?
- Message passing (e.g., MPI) or bulk synchronous processing (e.g., Spark) prevalent

Why did it fail?

Virtual Machine

Many papers in 1990s:

- Very compelling abstraction
- Many hard challenges, so many researchers worked on it

Today

- VMs everywhere
- Containers (e.g., docker) take this concept to the next level

Why did it succeed?

What are Hard/Fundamental Tradeoffs?

- Brewer's CAP conjecture: "Consistency, Availability, Partition-tolerance", you can have only 2/3 in a distributed system
- Tradeoff between latency and throughput for arbitrary updates in distributed systems, serving systems
 - Batch request to increase throughput, but hurts latency

Paper Presentation

Each student must present at least one paper

You must submit slides for review one week before presentation for feedback

- Please use Google slides
- On exception is next week (please send me the slides by Friday)

Lecture Format (starting with 1/27)

Provide context of the topic (5-10min)

Next, papers presented by you

- Two presentations of 15min each, followed by 20min discussion

Grading Policy

20% Class Participation

- Answer questions, join discussion, and present papers

20% Paper reviews

60% Project

Grading

Project: 60%

Paper blogs: 20%

Class participation: 20%

Exciting times in systems research

- New workloads, e.g., AI
- Huge scale → distributed computing
- Specialized hardware → heterogeneous computing
- New optimization techniques, e.g., AI
- ...

What is a good problem to work on?

Impact: someone cares about solving the problem

- The more people care, the better

Priority: you are among the first to work on the problem

- Increases the chances of being first to solve this problem
- Caveat: This is just “nice to have”. If you believe it’s an important problem and you are passionate about it, go for it!

Edge: You have an expertise (edge) to solve the problem

Incremental: can be partitioned into sub-problems

- Help you make rapid progress, iterate, better understand the problem

Some problems I worked on

<i>Problem</i>	<i>Impact</i> <i>(problem formulation)</i>	<i>Priority</i>	<i>Edge</i>	<i>Incremental</i>	<i>Comments</i>
Chord	Need for decentralized (no liability), efficient P2P system	Among several groups working on it	Experts in distributed systems and algorithms	Followed by many refinements	At some point, most cited paper 15,5K+ citations

Some problems I worked on

<i>Problem</i>	<i>Impact</i> <i>(problem formulation)</i>	<i>Priority</i>	<i>Edge</i>	<i>Incremental</i>	<i>Comments</i>
Chord	Need for decentralized (no liability), efficient P2P system	Among several groups working on it	Experts in distributed systems and algorithms	Followed by many refinements	At some point, most cited paper 15,5K+ citations
Spark	Need for fast, general-purpose big data analytics	Among first to see ML use cases (AMPLab) and interactive queries (industry; Conviva, FB)	Already experts in big data (in academia); started working on this 3 years earlier	Core Spark (3,000 LoC); Shark, Streaming, ML libs	High impact in industry 6.4K citations

Some problems I worked on

<i>Problem</i>	<i>Impact</i> <i>(problem formulation)</i>	<i>Priority</i>	<i>Edge</i>	<i>Incremental</i>	<i>Comments</i>
Chord	Need for decentralized (no liability), efficient P2P system	Among several groups working on it	Experts in distributed systems and algorithms	Followed by many refinements	At some point, most cited paper 15,5K+ citations
Spark	Need for fast, general-purpose big data analytics	Among first to see ML use cases (AMPLab) and interactive queries (industry; Conviva, FB)	Already experts in big data (in academia); started working on this 3 years earlier	Core Spark (3,000 LoC); Shark, Streaming, ML libs	High impact in industry 6,4K citations
Dominant Resource Fairness (DRF)	Need for better utilization for multi resource types workloads	Among first to see the problem of sharing the same cluster across multiple workloads (Mesos)	Experts in scheduling algorithms	Followed by other papers in different areas	Intellectual impact (microeconomy); implemented in a few systems 1,6K+ citations

Impact (1/2)

Do things that have not been done before



- “It always seems impossible until it is done” – Nelson Mandela
- Examples: zero-knowledge proof, beating chess/go world champion, outperform experts in medical diagnosis, ...

Do things much better than before (e.g., 10x)

- Examples: GPUs, Distributed Hash Tables (routing efficiency), in-memory databases, CNNs, ...

Impact (2/2)

Theory

- Improve space/time complexity of widely used algo (e.g., SGD)
- Extend existing techniques to new domains/environments, e.g., fair scheduler from network to CPU, distributed graph algorithms
- Prove new desirable properties for popular techniques (increase practitioners confidence in using them), e.g., show an allocation algo is strategy proof, show an algo is guaranteed to converge

Systems

- Support new workloads impractical before, e.g., big data processing
- Significantly improve efficiency/speed/cost, e.g., GPUs, dist algos
- “Democratize” technology, e.g., cloud computing, serverless, TensorFlow

Priority (1/2)

You experience problem first-hand

- One of the main reasons we are building systems/artifacts
 1. Build system solving a “need”
 2. Have people use it
 3. Understand things people want to do with your system you haven’t thought about, or you thought they can do but cannot
 4. Now you have a problem, you’ve “seen” first!

Take a bet on a new area and dive in

- Because it’s new area, few people are working on it
- This is one thing Berkeley has been excelling at, e.g.,
 - Complexity theory, approximate algorithms, databases, networking, sensor networks, OSes (UNIX), NOW, Big Data, ML/AI, ...

Priority (2/2)

Talk with people from other areas to find new problems, e.g.,

- Spark motivated by Matei's trying to solve Lester Mackey's to compete in \$1M Netflix's challenge

Berkeley's labs are set up to encourage you do exactly this!

"I notice that if you have the door to your office closed, you get more work done today and tomorrow, and you are more productive than most. But 10 years later somehow you don't know quite know what problems are worth working on; all the hard work you do is sort of tangential in importance. ... I can say there is a pretty good correlation between those who work with the doors open and those who ultimately do important things."

– Richard Hamming, Turing Award Winner

("You and Your Research": <http://www.cs.virginia.edu/~robins/YouAndYourResearch.html>)

Edge

If you do not have expertise to solve the problem, talk with people who do!

Again, labs like Sky Computing Lab are set up to encourage you do exactly this!

Incremental

Reach the first meaningful milestone fast

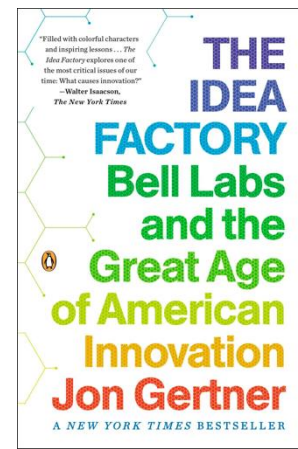
- Critical to get the project off the ground, maintain the excitement of the students and, if a system, start growing community
- Examples: Mesos, Spark, Ray provided a useful system within 1 year

Decompose project in a series of milestones providing increasing value

- Examples: Spark core → Shark (SQL) → Streaming → MLlib
- Developing a chess algorithm to first beat an amateur, then a master, then a grand master, and finally the world champion

Holly grail example: Bell Labs achieving its vision of “enabling any two people in the world communicate”

- Enable two people to talk by phone between Boston and NY
- Enable two people to talk by phone across US between NY and SF
- Enable two people to talk by phone across Atlantic between NY and London....
(see “The Idea Factory” book)



Questions to ask when picking a problem

What will happen if I solve this problem? Will anyone care? Will anything change in the way people do things?

- If you hope for a quantitative gain, simplify the problem and see improvement; if improvement not what you hoped for, reconsider

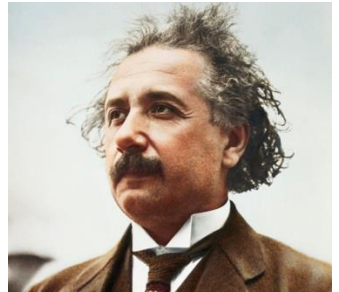
Who else is working on the problem? (Again, big caveat here; if you believe the problem is important and you can solve it go for it!)

Why me? Do I have any edge on solving this problem? Am I expert in some techniques I think it will help me solve it? Did I solve similar problems?

Is there a meaningful milestone I can get to fast?

“The mere formulation of a problem is far more often essential than its solution, which may be merely a matter of mathematical or experimental skill. To raise new questions, new possibilities, to regard old problems from a new angle requires creative imagination and marks real advances in science.”

Albert Einstein (1879 - 1955) Physicist & Nobel Laureate



The UNIX Time-Sharing System

Dennis M. Ritchie and Ken Thompson
Bell Laboratories

UNIX is a general-purpose, multi-user, interactive operating system for the Digital Equipment Corporation PDP-11/40 and 11/45 computers. It offers a number of features seldom found even in larger operating systems, including: (1) a hierarchical file system incorporating demountable volumes; (2) compatible file, device, and inter-process I/O; (3) the ability to initiate asynchronous processes; (4) system command language selectable on a per-user basis; and (5) over 100 subsystems including a dozen languages. This paper discusses the

1. Introduction

There have been three versions of UNIX. The earliest version (circa 1969–70) ran on the Digital Equipment Corporation PDP-7 and -9 computers. The second version ran on the unprotected PDP-11/20 computer. This paper describes only the PDP-11/40 and /45 [1] system since it is more modern and many of the differences between it and older UNIX systems result from redesign of features found to be deficient or lacking.

Since PDP-11 UNIX became operational in February 1971, about 40 installations have been put into service; they are generally smaller than the system described here. Most of them are engaged in applications such as the preparation and formatting of patent applications and other textual material, the collection and processing of trouble data from various switching machines within the Bell System, and recording and checking telephone service orders. Our own installation is used mainly for research in operating systems, languages, computer networks, and other topics in computer science, and also for document preparation.

Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for interactive use need not be expensive either in equipment or in human effort: UNIX can run on hardware costing as little as \$40,000, and less than two man years were spent on the main system software. Yet UNIX contains a number of features seldom offered even in much larger systems. It is hoped, however, the users of UNIX will find that the most important characteristics of the system are its simplicity, elegance, and ease of use.



PDP-11/40

The Unix Time-Sharing System

*“Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for interactive use need not be expensive either in equipment or in human effort: UNIX can run on hardware costing as little as \$40,000, and less than **two man years** were spent on the main system software.”*

Prior Approaches to the Problem

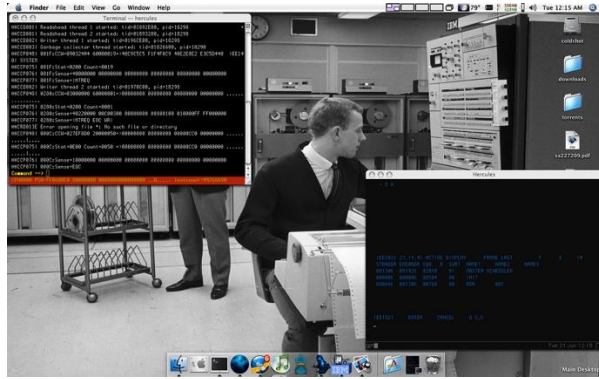
Batch Processing Systems:

- Users submitted jobs for sequential execution.
- No interactivity or real-time resource sharing.

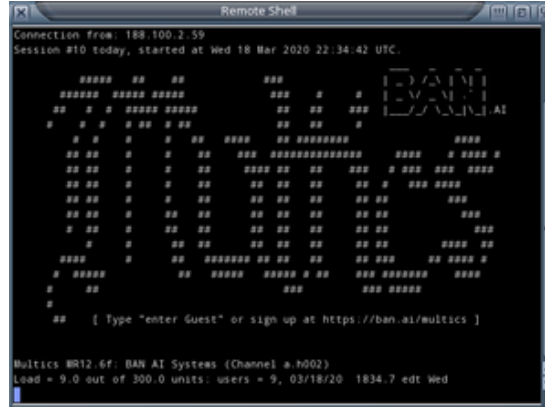
Time-Sharing Systems (e.g., CTSS, Multics):

- CTSS: Early interactive system but limited in scalability.
- Multics: Ambitious and powerful but complex, costly, and hardware-dependent.

The OS Landscape



IBM OS/360 (1966)
Batch-processing system



Multics (MIT, 1969)
Time-sharing system



CP/M (1974):
single user, single program

Main Ideas and Key Design Decisions

1. Simplicity:

- Small codebase to reduce complexity.
- Written in C to make it portable.

2. File System Design:

- Unified file structure (everything is a file).
- Hierarchical directory structure.

3. Process Model:

- Processes as first-class entities.
- Support for easy process creation and communication.

4. Shell as User Interface:

- Command-line shell for user interaction.
- Pipelines to chain programs for composability.

Key Design Decision 1 - Simplicity

- Minimalist kernel with essential functionality
- **Written in C**, unlike other systems written in assembly
 - Enable portability!
 - Faster to develop, change, iterate
 - Acceptable overhead (about 30% in code size)
- Focused on doing a few things well rather than being overly ambitious.

Key Design Decision 2 - Unified File System

Basically, all key features of today's file systems!!

- Hierarchical directories
- Mounted file systems (removable file systems)
- Links to files
- Access control and “super-user”

Files

Low level and flexible

- No predefined records; just bytes of block of bytes
- No predefined size; you write a new byte, the size increases

Everything is a file!

- Abstract away differences between devices and styles of access (e.g., sequential vs random)

Discussion: everything is a file

Devices are different:

- Disks: read/write, relatively high bandwidth
- Printers: only write, medium bandwidth
- Keyboard: only read, very low bandwidth
- Display: only write, low-to-high bandwidth

So why have the same abstraction/interface??

Discussion: why not locks?

How do you deal with:

- Different processes writing in the same file?
- One process reading while another writing?
- One process reading/writing while another process deletes it?

So why not locks?

- Context: System R (first relational database) was developed around the same time

Key Design Decision 3 - Process Management

- Image (current state of process) and processes (execution of an image)
- Lightweight process creation using `fork()`
- Parent-child process relationship simplifies process control
- Inter-process communication supported by pipes

Key Design Decision 4 - The Shell

- Interactive shell as a command-line interface
- Pipelining: Output of one program as input to another
(ls | grep)
- Encourages small, modular utilities for specialized tasks

Perspectives

1. Designed by programmers for programmers, i.e., easy to write, test, and run programs → time-sharing, interactive
2. Severe size constraints on the system and its software → encouraged economy and elegance in design
3. Dog-fooding: designers use the system → motivated them to correct the deficiency (encouraged by the fact that all source programs were always available and easily modified)

Key Results of UNIX

- Efficient use of hardware resources.
- Portability due to high-level language (C).
- Broad adoption across academic, research, and commercial environments.
- Foundation for many modern operating systems (e.g., Linux, macOS).

Limitations of the Paper

- Limited discussion on scalability to larger systems
- Sparse coverage of security and multi-user isolation
- Focused on small systems, not directly addressing high-availability or fault tolerance

Why Did This Paper Have an Impact?

- Introduced foundational OS principles still relevant today
- Demonstrated the power of simplicity and modularity
- Kickstarted the development of an ecosystem of UNIX-like systems
- Sparked widespread adoption in academia, leading to trained professionals familiar with UNIX