CS 6240: Parallel Data Processing in MapReduce

Mirek Riedewald

1

Course Information

- Homepage: http://www.ccs.neu.edu/home/mirek/classes/2011-F-CS6240/
  - Announcements
  - Lecture handouts
  - Office hours
- Homework management through Blackboard
- Prerequisites: CS 5800/CS 7800 ~~and CS 5600/CS 7600~~, or consent of instructor

2

Grading

- Homework/project: 40%
- Exams: Midterm 25%, Final 30%
- Participation: 5%
  - Prepare lecture notes, participate in class
- No copying or sharing of homework solutions!
  - But you can discuss general challenges and ideas
- Material allowed for exams
  - Any handwritten notes (originals, no photocopies)
  - Printouts of lecture summaries distributed by instructor
  - Nothing else

3

Instructor Information

- Instructor: Mirek Riedewald (332 WVH)
  - Office hours: Wed 4:30-5:30pm, Thu 11am-noon
  - Can email me your questions
  - Email for appointment if you cannot make it during office hours (or stop by for 1-minute questions)
- TA: no TA

4

Course Materials

- Hadoop: The Definitive Guide by Tom White
- Hadoop in Action by Chuck Lam
  - Both available from Safari Books Online at http://0-proquest.safaribooksonline.com.ilsprod.lib.neu.edu/
  - Use your myNEU credentials
- Other resources mentioned in syllabus and class homepage

5

Course Content and Objectives

- How to process massive amounts of data at large scale
  - Different from traditional approaches to parallel computation for smaller data
- Learn important fundamentals of selected approaches
  - Current trends and architectures
  - Coordinating multiple processes: mutual exclusion and consensus
  - Parallel programming in (raw) MapReduce
    - Programming model and Hadoop open source implementation
  - Creating data processing workflows with PigLatin
  - MapReduce versus SQL and other related approaches
- Many problem types and some design patterns

6

Course Content and Objectives

- Gain an intuition for how to deal with large-data problems
- Hands-on MapReduce practice
  - Writing MapReduce programs and running them on a cluster of machines
  - Understanding the system architecture and functionality below MapReduce
  - Learning about limitations of MapReduce
- Might produce publishable research

7

Words of Caution 1

- We can only cover a small part of the parallel computation universe
  - Do not expect all possible architectures, programming models, theoretical results, or vendors to be covered
  - Explore complementary courses in CCIS and ECE
- This really is an algorithms course, not a basic programming course
  - But you will need to do a lot of non-trivial programming

8

Words of Caution 2

- This is a new course, so expect rough edges like too slow/fast pace, uncertainty in homework load estimation
- There are few certain answers, as people in research and leading tech companies are trying to understand how to deal with BIG data
- We are working with cutting edge technology
  - Bugs, lack of documentation, Hadoop is changing API
  - Cluster might just go down, especially when everybody runs their programs 5 min before the deadline
- In short: you have to be able to deal with inevitable frustrations and plan your work accordingly…
- …but if you can do that and are willing to invest the time, it will be a rewarding experience

9

How to Succeed

- Attend the lectures and take your own notes
  - Helps remembering (compared to just listening)
  - Capture lecture content more individually than our handouts
  - Free preparation for exams
- Go over notes, handouts, book soon after lecture
  - Try to explain material to yourself or friend
- Look at content from previous lecture right before the next lecture to "page-in the context"

10

How to Succeed

- Ask questions during the lecture
  - Even seemingly simple questions show that you are thinking about the material and are genuinely interested in understanding it
- Work on the HW assignment as soon as it comes out
  - Can do most of the work on your own laptop
  - Time to ask questions and deal with unforeseen problems
  - We might not be able to answer all last-minute questions right before the deadline
- Students with disabilities: contact me by September 14

11

What Else to Expect?

- Need strong Java programming skills
  - Code for Hadoop open source MapReduce system is in Java
  - Hadoop supports other languages, but use at your own risk (we cannot help you and have not tested it)
- Need strong algorithms background
  - Analyze problems and solve them using unfamiliar tools like Map and Reduce functions
- Basic understanding of important system concepts
  - File system, processes, network basics, computer architecture

12

Why Focus on MapReduce?

- MapReduce is viewed as one of the biggest breakthroughs for processing massive amounts of data.
- It is widely used at technology leaders like Google, Yahoo, Facebook.
- It has huge support by the open source community.
  - Numerous active projects under Apache Hadoop
- Amazon provides special support for setting up Hadoop MapReduce clusters on its cloud infrastructure.
- It plays a major role in current database research conferences (and many other research communities)

13

Let us first look at some recent trends and developments that motivated MapReduce and other approaches to parallel data processing.

14

Why Parallel Processing?

- Answer 1: large data

15

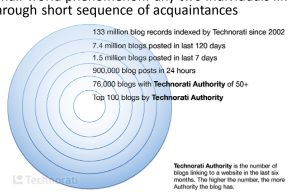
How Much Information?

- Source: http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/execsum.htm
- 5 exabytes (10^{18}) of new information from print, film, optical storage in 2002
  - 37,000 times Library of Congress book collections (17M books)
- New information on paper, film, magnetic and optical media doubled between 2000 and 2003
- Information that flows through electronic channels—telephone, radio, TV, Internet—contained 18 exabytes of new information in 2002

16

Web 2.0

- Billions of Web pages, social networks with millions of users, millions of blogs
  - How do friends affect my reviews, purchases, choice of friends
  - How does information spread?
  - What are "friendship patterns"
    - Small-world phenomenon: any two individuals likely to be connected through short sequence of acquaintances



17

Facebook Statistics

- Taken in 8/2011 from http://www.facebook.com/press/info.php?statistics
- 750M active users, 130 friends on average
- 900M objects (pages, groups, events, community pages)
- 30 billion pieces of content (web links, news stories, blog posts, notes, photo albums, etc.) shared each month
  - Avg. user creates 90 pieces of content per month

18

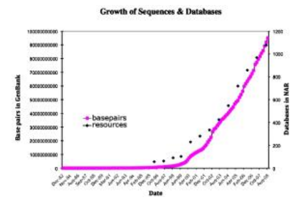
Business World

- Fraudulent/criminal transactions in bank accounts, credit cards, phone calls
  - Billions of transactions, real-time detection
- Retail stores
  - What products are people buying together?
  - What promotions will be most effective?
- Marketing
  - Which ads should be placed for which keyword query?
  - What are the key groups of customers and what defines each group?
- Spam filtering

19

eScience Examples

- Genome data
- Large Hadron Collider
  - Petabytes of raw data per year
- SkyServer
  - 818 GB, 3.4 billion rows
- DataONE
  - "Universal access to data about life on earth and the environment"
- Cornell Lab of Ornithology
  - 100M observations, 100s of attributes



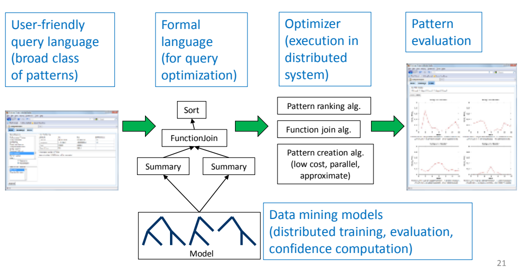
Source: Nature

20

Our Scolopax Project

Scolopax

- Search for patterns in prediction models based on user preferences
  Make this as easy and fast as Web search



21

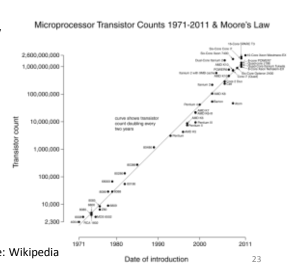
Why Parallel Processing?

- Answer 1: large data
- Answer 2: hardware trends

22

The Good Old Days

- Moore's Law: number of transistors that can be placed inexpensively on an integrated circuit doubles about every 2 years
- Computational capability improved at similar rate
  - Sequential programs became automatically faster
- Parallel computing never became mainstream
  - Reserved for high-performance computing niches

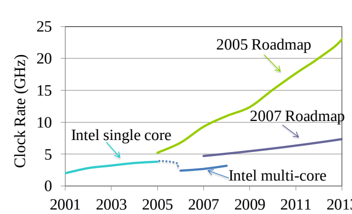


Source: Wikipedia

23

New Realities

- "Party" ended around 2004
- Heat issues prevent higher clock speeds
- Clock speed remains below 4 GHz



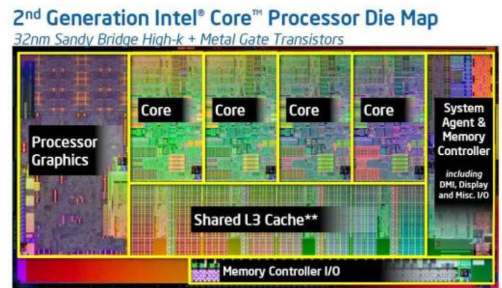
Source:
Dave Patterson, UCB

24

Multi-Core CPUs

- Clock speed stagnates, but number of cores increases
  - Core is like a processor, but shares chip with other cores
  - Cores typically share some cache, memory bus, access to same main memory
- Need to keep multiple cores busy to exploit additional transistors on chip
  - Multi-threaded applications

Processor Example (Source: Intel)



Die	Number of Transistors (mio)	Die size with Scribe (mm2)
4+2	995	216
2+2	624	149
2+1	504	131

** Cache is shared across all 4 cores and processor graphics

Typical Multi-Core Properties

- Each core has some local cache (e.g., L1, L2)
- The cores share some cache (e.g., L3)
- All cores access same memory through bus
- Misses become much more expensive from L1 to L3, even more when accessing memory

Important Numbers (Source: Google's Jeff Dean @LADIS'09)

L1 cache reference	0.5
Branch mispredict	5
L2 cache reference	7
Mutex lock/unlock	25
Main memory reference	100
Compress 1 KB with Zippy	3,000
Send 2 KB over 1 Gbps network	20,000
Read 1 MB sequentially from memory	250,000
Round trip within same data center	500,000
Disk seek	10,000,000
Read 1 MB sequentially from disk	20,000,000
Send packet CA -> Holland -> CA	150,000,000

All times in ns.

Other Trends

- Datacenter as a computer
  - Hundreds to tens of thousands of commodity machines for large-scale data processing
- Cloud computing
  - Often powered by data center
- GPU computing
  - Initially developed for fast parallel graphics computations, now also used for general computations
- Parallel data processing is becoming mainstream

Parallel Architectures

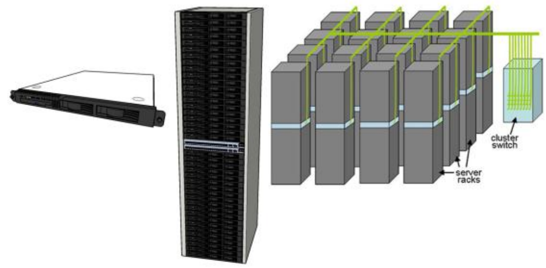
- Multi-core chips
- Datacenter as a computer

Warehouse-Scale Computer (WSC)

- Hundreds or thousands of commodity PCs
  - Better cost per unit of computational capability than specialized hardware due to economies of scale of commodity hardware
  - Easy to “scale out” by adding more machines
- Organized in racks in data centers
- Relatively homogenous hardware and system software platform with common system management layer
  - Often run smaller number of very large applications like Internet services

31

Basic Architecture



Source: Barroso, Holzle (2009)

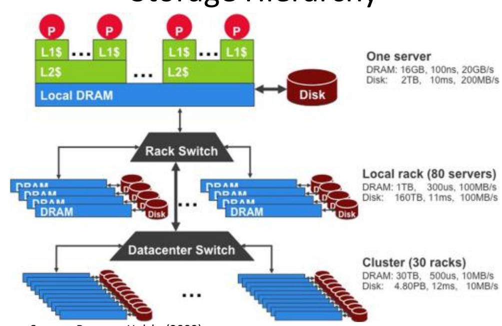
32

Typical Specs

- Low-end servers in 1U enclosure in 7’ rack
- Rack-level switch with 1- or 10-Gbps links
- Connected by one or more cluster switches
  - Can include >10,000 servers
- Local (cheap) disks on each server
  - Managed by global distributed file system
- Might have Network Attached Storage (NAS) devices for more centralized storage solution

33

Storage Hierarchy



Source: Barroso, Holzle (2009)

34

Programming WSCs

- Build cluster infrastructure and services that hide architecture complexity from developers
  - Program like a single big computer, but avoid inefficient code
- Need easy way to keep hundreds or thousands of CPUs busy
- Handle failures transparently
  - With 1000 commodity machines, failures are the norm, not the exception
  - Developers want to focus on their application, not how to deal with failures of hardware and low-level services
- This is where MapReduce comes into the picture!

35

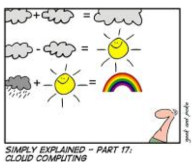
Parallel Architectures

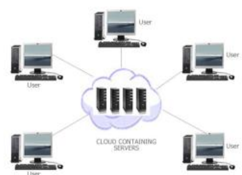
- Multi-core chips
- Datacenter as a computer
- Cloud computing

36

The Cloud

- Many different versions of Clouds
- Common idea: customers use virtual resources without knowing details about underlying hardware
  - Could run on cluster, multiple data centers, or large parallel machine
- Typical use 1: reserve virtual machines to create virtual cluster
- Typical use 2: connect through Web browser and run favorite application
- Typical use 3: build own app on top of services offered by Cloud provider
  - Database, document management, Web design, workflow, analytics





37

Cloud Computing

- Goal: Move data and programs from desktop PCs and corporate server rooms to "compute cloud"
- Related buzzwords: on-demand computing, software as a service (SaaS), Internet as platform
- Starts to replace shrink-wrap software
  - MSFT Word on desktop PC vs. Google Docs

38

Back to the Future…

- 1960s: service bureaus, time-sharing systems
  - Hub-and-spoke configuration: terminal access through phone lines, central site for computation
- 1980s: PCs "liberate" programs and data from central computing center
  - Customization of computing environment
  - Client-server model

39

…or not?

- Cloud is not the same as 1960's hub
  - Client can communicate with many servers at the same time
  - Servers can communicate with each other
- Still, functions migrate to distant data centers
  - "Core" and "fringe"
    - Storage, computing, high bandwidth, and careful resource management in core
    - End users initiate requests from fringe

40

Why Clouds?

- High price of total control
  - Software installation, configuration, and maintenance
  - Maintenance of computing infrastructure
  - Difficult to grow and shrink capacity on demand
- Easier software development
  - Replaces huge variety of operating environments by computing platform of vendor's choosing
  - But: server interaction with variety of clients
- Easier to deploy updates and bug fixes
- Easier to leverage multi-core, parallel systems
  - Single instance of Word cannot utilize 100 cores, but 100 instances of Word can

41

Example Cloud Offerings

- Document processing
  - Google Docs: word processor, spreadsheet, presentations
  - Adobe: Acrobat.com, Photoshop Express
  - Microsoft Office 365
- Enterprise applications
  - Salesforce.com: customer relationship management, sales marketing apps
  - Microsoft Dynamics CRM, IBM Tivoli Live
- Cloud infrastructure
  - Amazon Web Services: storage, computing as needed (pay as you go)
  - IBM Smart Cloud, Google App Engine, Force.com, Microsoft Azure
- Cloud OS
  - User interface in Web browser
  - New browser wars: browser as new Cloud OS

42

Challenges

- Scalability
  - More users, complex interactions between applications
- Many-to-many communication
  - Client invokes programs on multiple servers, server talks to multiple clients
- Browser is limited compared to traditional OS
  - Limited functionality
  - Fewer development tools

43

More Challenges

- Heterogeneous environment
  - Database backend with SQL
  - JavaScript, HTML at client
  - Server app written in PHP, Java, Python
  - Information exchanged as XML
- New role for open source movement?
  - Open source word processor vs. running a service

44

Biggest Problems

- Privacy, security, reliability
  - What if the service is not accessible?
  - Who owns the data?
  - Lose access to data if bill not paid?
  - Guarantee that deleted documents are really gone?
  - How aggressive about protecting data, e.g., against government access?
  - How to know if data is leaked to third party?

45

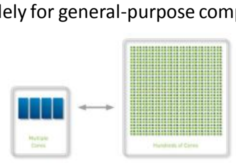
Parallel Architectures

- Multi-core chips
- Datacenter as a computer
- Cloud computing
- GPU computing

46

GPU vs. CPU

- Optimized for massively parallel processing
  - Graphics processing
- Challenge: how to create applications for 100s of cores?
  - NVIDIA developed CUDA
  - Used widely for general-purpose computations

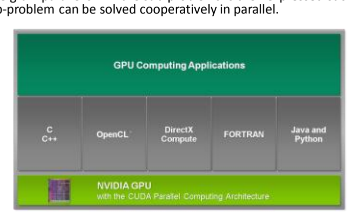


Source: NVIDIA

47

CUDA (Source: NVIDIA)

- CUDA programming model provides abstractions for data and task parallelism
  - Programmer can express parallelism in high-level languages such as C, C++, Fortran or driver APIs such as OpenCL™ and DirectX™-11 Compute
  - Programming model guides programmers to partition the problem into coarse sub-problems that can be solved independently in parallel
  - Fine grain parallelism in the sub-problems is then expressed such that each sub-problem can be solved cooperatively in parallel.



48

Course Content in a Nutshell

- In large-scale data processing, usually the same computation needs to be applied to a lot of data
  - Possibly many such steps (think "workflow")
- Divide the work between multiple processors
  - Make sure you can handle data transfer efficiently
- Combine intermediate results from multiple processors

49

Why This Is Not So Easy

- How can the work be partitioned?
- What if too much intermediate data is produced?
- How do we start up and manage 1000s of jobs?
- How do we get large data sets to processors or move processing to the data?
- How do we deal with slow responses and failures?

50

More Problems

- Shared resources limit scalability
  - Cost of managing concurrent access
- Shared-nothing architectures still need communication for processes to share data
- Easy to get into problems like deadlocks and race conditions
- It is generally difficult to reason about the behavior and correctness of concurrent processes
  - Especially when failures are part of the model
- Inherent tradeoff between consistency, availability, and partition tolerance (Brewer's Conjecture)
- In short: writing parallel programs is HARD.

51

What Can We Do?

- Work at the right level of abstraction
  - Too low-level: difficult to write programs, e.g., to deal with locks; need to customize code for different systems
  - Too high-level: poor performance if control for crucial bottleneck is "abstracted away"
- Use more declarative style of programming
  - Define WHAT needs to be computed, not HOW this is done at the low level
  - Well-known success story: SQL and databases

52

Recipes for Success

- Use hardware that can scale out, not just up
  - Doubling the number of commodity servers is easy, but buying a double-sized SMP machine is not.
- Have data located near the processors
  - Sending petabytes around is not a good idea
- Avoid centralized resources that are likely bottlenecks, e.g., single shared memory bus for many cores
- Read and write data sequentially
  - Assume random I/O takes 20 msec, disk streams data sequentially at 100 MB/sec, and record size is 1 KB
  - During 1 random I/O, can read 2000 records sequentially
- MapReduce does all this, and its level of abstraction seems to have hit a sweet spot

53

Algorithms First

- No matter which parallel programming model we use, we first need to understand what part of a computation can be performed in parallel
- More precisely…

54

Writing Parallel Programs

- Analyze problem and identify what can be done in parallel
  - Dependencies: if I need data D as input for a task, then I cannot run this task and the creation of D in parallel.
  - Coordination requirements: when do parallel tasks have to communicate and how much data is sent?
  - Best sequential algorithm might not be easy to parallelize—find alternative solutions
- Create an efficient implementation
  - Make sure solution is a good fit for the given architecture and programming model

55

Examples

- Let us look at some examples to get a feeling for the challenges

56

Sum Of Integers

- Compute sum of a large set of integers
- Sequential: simple for-loop (scan)
- Parallel: assign chunk of data to each processor to compute local sum, then add them together

- Algorithmically easy, but…
  - Where do the chunks come from? Partitioning data file into multiple chunks might take as long as sequential computation.
  - What if data transfer is the bottleneck? Then pushing k chunks from disk to k cores might not be possible in parallel.
  - Who computes final sum and how do the local sums get there?

57

Word Count

- Count the number of occurrences of each word in a large document
- Sequential: read document sequentially, update counters for each word
  - Need data structure, e.g., hash map, to keep track of counts
- Parallel: each processor does this for a chunk using local data structure, then counts are aggregated
- Improvement (?): use shared data structure for counts
  - Good: no "replication", no need for final summation step
  - Bad: need to coordinate access to shared data structure, not a good fit for shared-nothing architecture
- What if some documents are much larger than others?
  - Ned to deal with data skew, e.g., break up large documents

58

Equi-Join

- Given two data sets $S=(s_1,s_2,\ldots)$ and $T=(t_1,t_2,\ldots)$ of integers, find all pairs (s_i,t_j) where $s_i=t_j$
- Common operation in database systems
- Many sequential algorithms
  - Nested loops, symmetric hash, index nested loops, sort merge, partition-based

- How to parallelize this?

59

Parallel Index Nested Loops

- For each tuple s_i in S
  - Look up all matching t_j in index(T) and output (s_i,t_j)
- Can partition S into chunks S_1 and S_2
- But each processor needs access to entire index(T)
  - Copy index(T) to each processor, which is not so great if index(T) is very large
  - Use shared index: if read-only, no need for mutual exclusion, but access latency for remote machines kills lookup performance
- Fancier approach: partition index(T)
  - Tricky: non-uniform access pattern (e.g., root vs. leaf) and possibly high communication cost

60