Quantitative Big Imaging

Scaling up / Big Data

ETHZ: 227-0966-00L

```
## Loading required package: knitr
```

Course Outline

- 25th February - Introduction and Workflows
- 3rd March - Image Enhancement (A. Kaestner)
- 10th March - Basic Segmentation, Discrete Binary Structures
- 17th March - Advanced Segmentation
- 24th March - Analyzing Single Objects
- 7th April - Analyzing Complex Objects
- 14th April - Many Objects and Distributions
- 21st April - Statistics and Reproducibility
- 28th April - Dynamic Experiments
- 12th May - Scaling Up / Big Data
- 19th May - Guest Lecture - High Content Screening
- 26th May - Guest Lecture - Machine Learning / Deep Learning and More Advanced Approaches
- 2nd June - Project Presentations

Literature / Useful References

Big Data

- Google’s Presentation on Distributed Computing
- Slides
- MapReduce Paper: Jeffrey Dean, et al. (n.d.). MapReduce: Simplified Data Processing on Large Clusters.
- Scalable Systems Course
- Tutorial in Hadoop
- [Intro to Data Science @UCB] (http://amplab.github.io/datascience-sp14/)

Cluster Computing

- Altintas, I. (2013). Workflow-driven programming paradigms for distributed analysis of biological big data. In 2013 IEEE 3rd International Conference on Computational Advances in Bio and medical Sciences (ICCABS) (pp. 1–1). IEEE. doi:10.1109/ICCABS.2013.6629243
- Condor High-throughput Computing
- Condor Setup at ITET
- Sun (now Oracle) Grid Engine or here

Databases

- Ollion, J., Cochennec, J., Loll, F., Escudé, C., & Boudier, T. (2013). TANGO: a generic tool for high-throughput 3D image analysis for studying nuclear organization. Bioinformatics (Oxford, England), 29(14), 1840–1. doi:10.1093/bioinformatics/btt276

Cloud Computing

- Amazon S3
- Sitaram, D., & Manjunath, G. (2012). Moving To The Cloud. null (Vol. null). Elsevier. doi:10.1016/B978-1-59749-725-1.00006-8
- Duan, P., Wang, W., Zhang, W., Gong, F., Zhang, P., & Rao, Y. (2013). Food Image Recognition Using Pervasive Cloud Computing. In 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing (pp. 1631–1637). IEEE. doi:10.1109/GreenCom-iThings-CPSCom.2013.296

Outline

- Motivation
- Computer Science Principles
- Parallelism
- Distributed Computing
- Imperative Programming
- Declarative Programming
- Organization
- Queue Systems / Cluster Computing
- Parameterization
- Databases
- Big Data
- MapReduce
- Spark
- Streaming
- Cloud Computing
- Beyond / The future

Motivation

There are three different types of problems that we will run into.

Really big data sets

- Several copies of the dataset need to be in memory for processing
- Computers with more 256GB are expensive and difficult to find
- Even they have 16 cores so still 16GB per CPU
- Drive speed / network file access becomes a limiting factor
- If it crashes you **lose** everything
- or you have to manually write a bunch of mess check-pointing code

Many datasets

- For genome-scale studies 1000s of samples need to be analyzed identically
- Dynamic experiments can have hundreds of measurements
- Animal phenotyping can have many huge data-sets (1000s of 328GB datasets)
- Radiologist in Switzerland alone make 1 Petabyte of scans per year

Exploratory Studies

- Not sure what we are looking for

- Easy to develop new analyses
- Quick to test hypotheses

Example Projects

Zebra fish Full Animal Phenotyping

Full adult animal at cellular resolution 1000s of samples of full adult animals, imaged at 0.74 μm resolution: Images 11500 x 2800 x 628 $\longrightarrow$ 20-40GVx / sample

- Identification of single cells (no down-sampling)
- Cell networks and connectivity
- Classification of cell type
- Registration with histology

Brain Project

Whole brain with cellular resolution 1 cm^3 scanned at 1 μm resolution: Images $\longrightarrow$ 1000 GVx / sample

- Registration separate scans together
- Blood vessel structure and networks
- Registration with fMRI, histology

What is wrong with usual approaches?

Normally when problems are approached they are solved for a single task as quickly as possible - I need to filter my image with a median filter with a neighborhood of 5 x 5 and a square kernel - then make a threshold of 10 - label the components - then count how many voxels are in each component - save it to a file

```
im_in=imread('test.jpg');
im_filter=medfilt2(im_in,[5,5]);
cl_img=bwlabel(im_filter>10);
cl_count=hist(cl_img,1:100);
dlmwrite(cl_count,'out.txt')
```

- What if you want to compare Gaussian and Median?
- What if you want to look at 3D instead of 2D images?
- What if you want to run the same analysis for a folder of images?

You have to rewrite everything, everytime

If you start with a bad approach, it is very difficult to fix, big data and reproducibility must be considered from the beginning

Computer Science Principles

Disclosure : There are entire courses / PhD thesis's / Companies about this, so this is just a quick introduction

- Parallelism
- Distributed Computing
- Resource Contention
- Shared-Memory
- Race Conditions
- Synchronization
- Dead lock
- Imperative
- Declarative

What is parallelism?

Parallelism is when you can divide a task into separate pieces which can then be worked on at the same time.

For example

- if you have to walk 5 minutes and talk on the phone for 5 minutes
- you can perform the tasks serially which then takes 10 minutes
- you can perform the tasks in parallel which then takes 5 minutes

Some tasks are easy to parallelize while others are very difficult. Rather than focusing on programming, real-life examples are good indicators of difficultly.

What is distributed computing?

Distributed computing is very similar to parallel computing, but a bit more particular. Parallel means you process many tasks at the same time, while distributed means you are no longer on the same CPU, process, or even on the same machine.

The distributed has some important implications since once you are no longer on the same machine the number of variables like network delay, file system issues, and other users becomes a major problem.

Distributed Computing Examples

1. You have 10 friends who collectively know all the capital cities of the world.

- To find the capital of a single country you just yell the country and wait for someone to respond (+++)
- To find who knows the most countries, each, in turn, yells out how many countries they know and you select the highest (++)

1. Each friend has some money with them

- To find the total amount of money you tell each person to tell you how much money they have and you add it together (+)
- To find the **median** coin value, you ask each friend to tell you you all the coins they have and you make one master list and then find the median coin (-)

Resource Contention

The largest issue with parallel / distributed tasks is the need to access the same resources at the same time

- memory / files
- pieces of information
- network resources

Dead-lock

Dining Philopher's Problem - 6 philosophers at the table - 6 forks - Everyone needs two forks to eat - Each philospher takes the fork on his left

Parallel Challenges

Coordination

Parallel computing requires a significant of coordinating between computers for non-easily parallelizable tasks.

Mutability

The second major issue is mutability, if you have two cores / computers trying to write the same information at the same it is no longer deterministic (not good)

Blocking

The simple act of taking turns and waiting for every independent process to take its turn can completely negate the benefits of parallel computing

Distributed Challenges

Inherits all of the problems of parallel programming with a whole variety of new issues.

Sending Instructions / Data Afar

Fault Tolerance

If you have 1000 computers working on solving a problem and one fails, you do not want your whole job to crash

Data Storage

How can you access and process data from many different computers quickly without very expensive infrastructure

Imperative Programming

Directly coordinating tasks on a computer.

- Languages like C, C++, Java, Matlab
- Exact orders are given (implicit time ordering)
- Data management is manually controlled
- Job and task scheduling is manual
- Potential to tweak and optimize performance

Making a soup

1. Buy vegetables at market
2. *then* Buy meat at butcher
3. *then* Chop carrots into pieces
4. *then* Chop potatos into pieces
5. *then* Heat water
6. *then* Wait until boiling then add chopped vegetables
7. *then* Wait 5 minutes and add meat

Declarative

- Languages like SQL, Erlang, Haskell, Scala, Python, R can be declarative
- Goals are stated rather than specific details
- Data is automatically managed and copied
- Scheduling is automatic but not always *efficient*

Making a soup

- Buy vegetables at market $\rightarrow shop_{veggies}$
- Buy meat at butcher $\rightarrow shop_{meat}$
- Wait for $shop_{veggies}$: Chop carrots into pieces $\rightarrow chopped_{carrots}$
- Wait for $shop_{veggies}$: Chop potatos into pieces $\rightarrow chopped_{potatos}$
- Heat water
- Wait for $boilingwater, chopped_{carrots}, chopped_{potatos}$: Add chopped vegetables
- Wait 5 minutes and add meat

Comparison

They look fairly similar, so what is the difference? The second is needlessly complicated for one person, but what if you have a team, how can several people make an imparitive soup faster (chopping vegetables together?)

Imperative soup

1. Buy {carrots, peas, tomatoes} at market
2. *then* Buy meat at butcher
3. *then* Chop carrots into pieces
4. *then* Chop potatos into pieces
5. *then* Heat water
6. *then* Wait until boiling then add chopped vegetables
7. *then* Wait 5 minutes and add meat

How can many people make a declarative soup faster? Give everyone a different task (not completely efficient since some tasks have to wait on others)

Declarative soup

- Buy {carrots, peas, tomatoes} at market $\rightarrow shop_{veggies}$
- Buy meat at butcher $\rightarrow shop_{meat}$
- Wait for $shop_{veggies}$: Chop carrots into pieces $\rightarrow chopped_{carrots}$
- Wait for $shop_{veggies}$: Chop potatos into pieces $\rightarrow chopped_{potatos}$
- Heat water

- Wait for $boiling_{water}, chopped_{carrots}, chopped_{potatos}$: Add chopped vegetables
- Wait 5 minutes and add meat

Results

Imperative

- optimize specific tasks (chopping vegetables, mixing) so that many people can do it faster
- Matlab/Python do this with fast-fourier-transforms (automatically uses many cores to compute faster)
- make many soups at the same time (independent)
- This leads us to cluster-based computing

Declarative

- run everything at once
- each core (computer) takes a task and runs it
- execution order does not matter
- wait for portions to be available (dependency)

Lazy Evaluation

- do not run anything at all
- until something needs to be exported or saved
- run only the tasks that are needed for the final result
- never buy tomatoes since they are not in the final soup

Organization

One of the major challenges of scaling up experiments and analysis is keeping all of the results organized in a clear manner. As we have seen in the last lectures, many of the results produced many text files - many files are difficult to organize - Matlab, R are designed for in-memory computation - Datasets can have many parameters and be complicated - Transitioning from Excel to Matlab or R means rewriting everything

Queue Computing

Queue processing systems (like Sun Grid Engine, Oracle Grid Engine, Apple XGrid, Condor, etc) are used to manage - resources (computers, memory, storage) - jobs (tasks to be run) - users Based on a set of rules for how to share the resources to the users to run tasks.

Resources

- a collection of processors (CPU and GPU)
- memory, local storage
- access to bandwidth or special resource like a printer
- for a given period of time

Jobs

- specific task to run
- necessary (minimal/maximal) resources to run with
- including execution time

Users

- accounts submitting jobs
- it can be undesirable for one user to dominate all of the resources all the time

Structure of Cluster

Master (or Name) Node(s)

The node with which every other node communicates, the main address.

Worker Nodes

The nodes where the computation is performed.

Scheduler

The actual process that decides which jobs will run using which resources (worker nodes, memory, bandwidth) at which time

Databases

A database is a collection of data stored in the format of tables: a number of columns and rows.

Animals

Here we have an table of the animals measured in an experiment and their weight

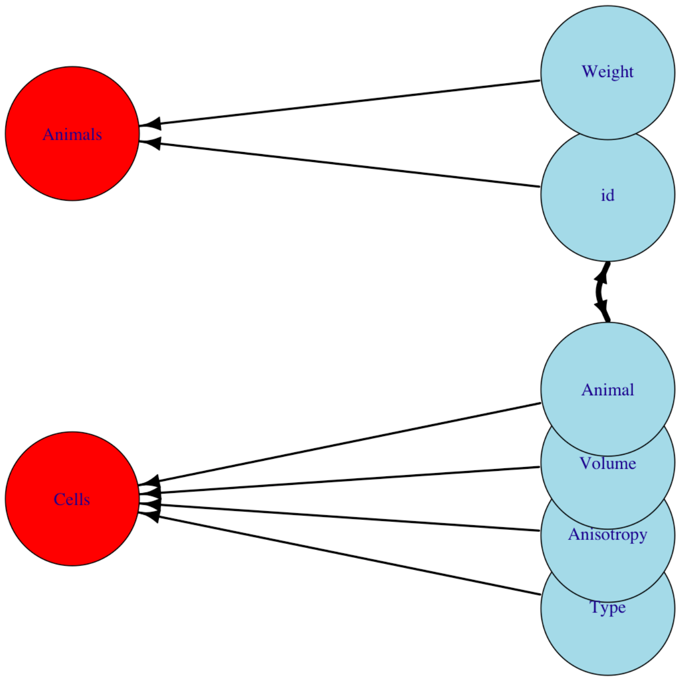
id	Weight
1	100
2	40
3	80

Cells

The cells is then an analysis looking at the cellular structures

Animal	Type	Anisotropy	Volume
1	Cancer	0.5	1.00
1	Healthy	1.0	2.00
2	Cancer	0.5	0.95

Relational-databases can store relationships between different tables so the relationship between *Animal* in table **Cells** and *id* in table **Animals** can be preserved.


Weight
Animals
id
Animal
Cells
Volume
Anisotropy
Type

SQL

SQL (pronounced Sequel) stands for structured query language and is *nearly* universal for both searching (called querying) and adding (called inserting) data into databases. SQL is used in various forms from Firefox storing its preferences locally (using SQLite) to Facebook storing some of its user information (MySQL and Hive). So refering to the two tables we defined in the last entry, we can use SQL to get information about the tables independently of how they are stored (a single machine, a supercomputer, or in the cloud)

Basic queries

- Get the volume of all cells

```
SELECT Volume FROM Cells
```

$$\rightarrow \left[1, 2, 0.95\right]$$

- Get the average volume of all cancer cells

```
SELECT AVG(Volume) FROM Cells WHERE Type = "Canc
```

$$\rightarrow 0.975$$

We could have done these easily without SQL using Excel, Matlab or R

More Advanced SQL

- Get the volume of all cells in heavy mice

```
SELECT Volume FROM Cells WHERE Animal IN
  (SELECT id FROM Animal WHERE Weight>80)
```

- Get weight and average cell volume for all mice

```
SELECT ATable.Weight,CTable.Volume FROM Animals as ATable
  INNER JOIN Cells as CTable on (ATable.id=CTable.Animal)
```

$$\rightarrow \left[1, 0.95 \right]$$

Networks using SQL

If we expand our SQL example to cellular networks with an additional table explicitly describing the links between cells

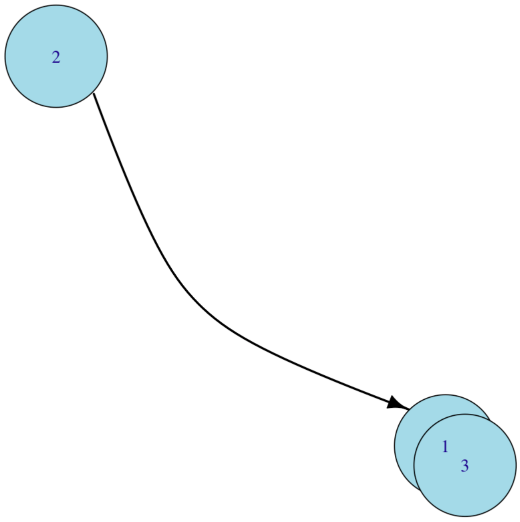


Table Representation

id1	id2	No.Juncs
1	2	800
1	3	40
3	1	300

Now to calculate how many connections each cell has

```
SELECT id,COUNT(*) AS connection_count FROM Cells
  INNER JOIN Network ON (id=id1 OR id=id2)
```

$$\rightarrow \begin{bmatrix} (1 & 3) \\ (2 & 1) \\ (3 & 2) \end{bmatrix}$$

Beyond SQL: NoSQL

Basic networks can be entered and queries using SQL but relatively simple sounding requests can get complicated very quickly

How many cells are within two connections of each cell

```
SELECT id,COUNT(*) AS connection_count FROM Cells as CellsA
  INNER JOIN Network as NetA ON Where (id=NetA.id1)
  INNER JOIN Network as NetB ON Where (NetA.id2=NetB.id1)
```

This is *still* readable but becomes very cumbersome quickly and difficult to manage

NoSQL (Not Only SQL)

A new generation of database software which extends the functionality of SQL to allow for more scalability (MongoDB) or specificity for problems like networks or graphs called generally **Graph Databases**

Big Data: Definition

Velocity, Volume, Variety

When a ton of heterogeneous is coming in fast.

Performant, scalable, and flexible

When scaling isn't scary

10X, 100X, 1000X is the same amount of effort

When you are starving for enough data

Director of AMPLab said their rate limiting factor is always enough interesting data

O 'clicks' per sample

A brief oversimplified story

Google ran into ‘big data’ and its associated problems years ago: Peta- and exabytes of websites to collect and make sense of. Google uses an algorithm called PageRank(tm) for evaluating the quality of websites. They could have probably used existing tools if page rank were some magic program that could read and determine the quality of a site

```
for every_site_on_internet
  current_site.rank=secret_pagerank_function(current_site)
end
```

Just divide all the websites into a bunch of groups and have each computer run a group, **easy!**

PageRank

While the actual internals of PageRank are not public, the general idea is that sites are ranked based on how many sites link to them

```
for current_site in every_site_on_internet
  current_pagerank = new SecretPageRankObj(current_site);
  for other_site in every_site_on_internet
    if current_site is_linked_to other_site
      current_pagerank.add_site(other_site);
    end
  end
  current_site.rank=current_pagerank.rank();
end
```

How do you divide this task? - Maybe try and divide the sites up: english_sites, chinese_sites, ... - Run pagerank and run them separately. - What happens when a chinese_site links to an english_site? - Buy a really big, really fast computer? - On the most-powerful computer in the world, one loop would take months

It gets better

- What happens if one computer / hard-drive crashes?
- Have a backup computer replace it (A backup computer for every single system)
- With a few computers ok, with hundreds of thousands of computers?
- What if there is an earthquake and all the computers go down?
- PageRank doesn’t just count
- Uses the old rankings for that page
- Run pagerank many times until the ranks converge

Google's Solution: MapReduce (part of it)

~~some people claim to have had the idea before, Google is certainly the first to do it at scale~~

Several engineers at Google recognized common elements in many of the tasks being performed. They then proceeded to divide all tasks into two classes **Map** and **Reduce**

Map

Map is where a function is applied to every element in the list and the function depends only on exactly that element

$$\vec{L} = \left[1, 2, 3, 4, 5\right]$$

$$f(x) = x^2$$

$$map(f \rightarrow \vec{L}) = \left[1, 4, 9, 16, 25\right]$$

Reduce

Reduce is more complicated and involves aggregating a number of different elements and summarizing them. For example the Σ function can be written as a reduce function

$$\vec{L} = \left[1, 2, 3, 4, 5\right]$$

$$g(a, b) = a + b$$

Reduce then applies the function to the first two elements, and then to the result of the first two with the third and so on until all the elements are done

$$reduce(f \rightarrow \vec{L}) = g(g(g(g(1, 2), 3), 4), 5)$$

MapReduce

They designed a framework for handling distributing and running these types of jobs on clusters. So for each job a dataset ($\vec{L}$), Map-task (f), a grouping, and Reduce-task (g) are specified

- Partition input data ($\vec{L}$) into chunks across all machines in the cluster

$$\downarrow$$

- Apply **Map** (f) to each element

$$\downarrow$$

- Shuffle and Repartition or Group Data

$$\downarrow$$

- Apply **Reduce** (g) to each group

$$\downarrow$$

- Collect all of the results and write to disk

All of the steps in between can be written once in a robust, safe manner and then used for every task which can be described using this MapReduce paradigm. These tasks $\langle \vec{L}, f(x), g(a,b) \rangle$ is refered to as a job.

Key-Value Pairs / Grouping

The initial job was very basic, for more complicated jobs, a new notion of Key-value (KV) pairs must be introduced. A KV pair is made up of a key and value. A key must be comparable / hashable (a number, string, immutable list of numbers, etc) and is used for grouping data. The value is the associated information to this key.

Counting Words

Using MapReduce on a folder full of text-documents. -

$$\vec{L} = \left[\text{" Info } \cdots \text{ ", " Expenses } \cdots \text{ "}, \cdots \right]$$

- **Map** is then a function f which takes in a long string and returns a list of all of the words (text seperated by spaces) as key-value pairs with the value being the number of times that word appeared - `f(x) = [(word,1) for word in  x.split(" ")]` - Grouping is then performed by keys (group all words together) - **Reduce** adds up the values for each word

```
L = ["cat dog car",
  "dog car dog"]
```

$$\downarrow \ \textbf{Map} \ : f(x)$$

```
[("cat",1),("dog",1),("car",1),("dog",1),("car",1),("dog",1)]
```

$$\downarrow \ \text{Shuffle / Group}$$

```
"cat": (1)
"dog": (1,1,1)
"car": (1,1)
```

$$\downarrow \ \textbf{Reduce} \ : g(a, b)$$

```
[("cat",1),("dog",3),("car",2)]
```

Hadoop

Hadoop is the opensource version of MapReduce developed by Yahoo and released as an Apache project. It provides underlying infrastructure and filesystem that handles storing and distributing data so each machine stores some of the data locally and processing jobs run where the data is stored. - Non-local data is copied over the network. - Storage is automatically expanded with processing power. - It’s how Amazon, Microsoft, Yahoo, Facebook, … deal with exabytes of data

Spark / Resilient Distributed Datasets

Technical Specifications

- Developed by the Algorithms, Machines, and People Lab at UC Berkeley in 2012
- General tool for all Directed Acyclical Graph (DAG) workflows
- Course-grained processing → simple operations applied to entire sets
- Map, reduce, join, group by, fold, foreach, filter,…
- In-memory caching

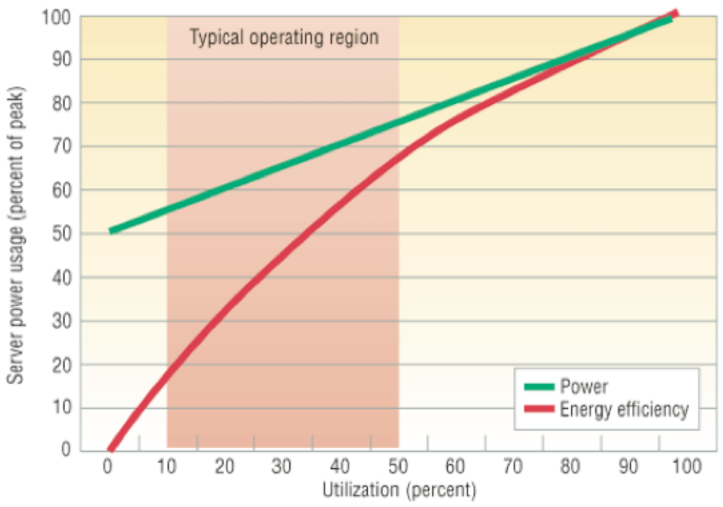
Zaharia, M., et. al (2012). Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing

Practical Specification

- Distributed, parallel computing without **logistics**, libraries, or compiling
- Declarative rather than imperative
- Apply operation f to each image / block
- **NOT** tell computer 3 to wait for an image from computer 2 to and perform operation f and send it to computer 1
- Even scheduling is handled automatically
- Results can be stored in memory, on disk, redundant or not

Cloud Computing

- Local resources are expensive and underutilized
- Management and updates are expensive and require dedicated IT staff



Cloud Resources

- Automatically setup
- Unlimited potential capacity and storage
- Cluster management already setup
- Common tools with many people using the same

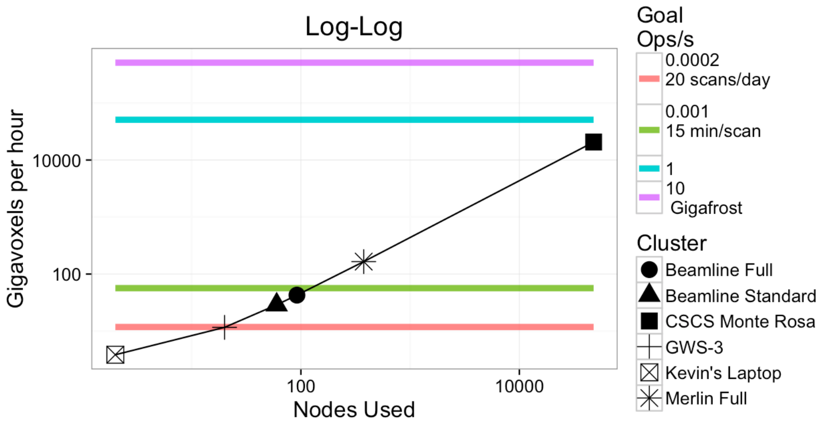
Near Future Imaging Goals

Needs

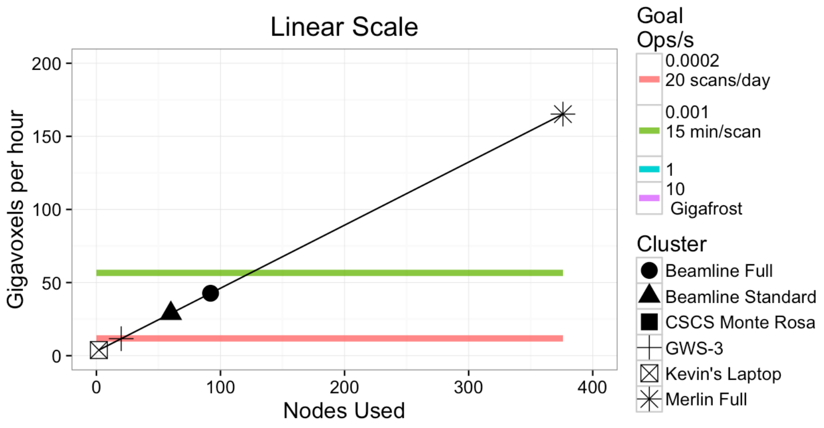
- Scale up to 1000GVx samples (eventually)
- Analyze standard measurements 14GVx regularly in a day
- Analyze as fast as we usually measure 15 minutes / scan

Would be nice

- Analyze scans as fast as we measure now (1 scan / minute)
- Analyze scans as fast as we could measure with Gigafrost (10 scans / minute)



Tomcat Goals



Tomcat Goals

Post-processing: Statistical Analysis

- Exploring phenotypes of 55 million cells.
- Our current approach has been either
- Group and divide then average the results.
- Average Cell Volume, etc.
- Average density or connectivity
- Detailed analysis of individual samples
- K-means clustering for different regions in bone / layers
- Principal component analysis for phenotypes

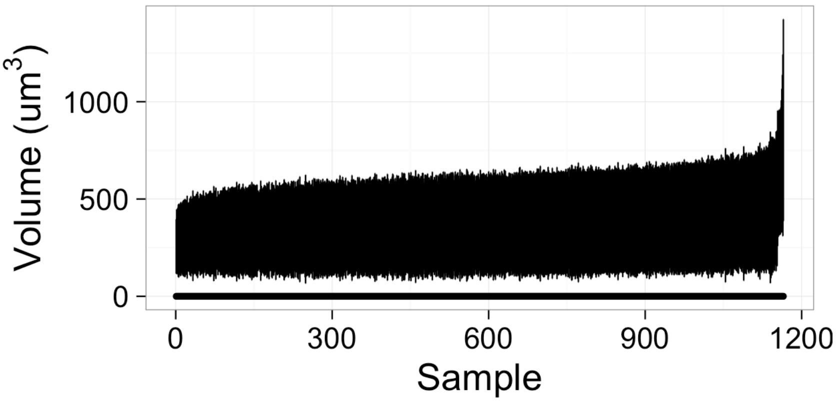
If we do that, we miss a lot!

Phenotype	**Within**	**Between**	**Ratio (%)**
Length	36.97286	4.278853	864.08349
Width	27.92322	4.733648	589.88801
Height	25.15456	4.636371	542.54855
Volume	67.85303	12.478975	543.73881
Nearest Canal Distance	70.35092	333.395326	21.10135
Density (Lc.Te.V)	144.40110	27.657829	522.09844
Nearest Neighbors (Lc.DN)	31.86131	1.835211	1736.11101
Stretch (Lc.St)	13.98019	2.359673	592.46289

Oblateness (Lc.Ob)	141.26874	18.464618	765.07808
The results in the ta	ble show the	within and b	etween sample variation for selected phenotypes in the first two columns and the ratio of the within and between sample numbers

Visualizing the Variation

How does this result look visually? Each line shows the mean ± standard deviation for sample. The range within a single sample is clearly much larger than between



Partial Volume Effect

Do not condense

With the ability to scale to millions of samples there is no need to condense. We can analyze the entire dataset in real-time and try and identify groups or trends within the whole data instead of just precalculated views of it.

Practical

1276 comma separated text files with 56 columns of data and 15-60K rows

Task	Single Core Time	Spark Time (40 cores)
Load and Preprocess	360 minutes	10 minutes
Single Column Average	4.6s	400ms
1 K-means Iteration	2 minutes	1s

Can iteratively explore and hypothesis test with data quickly

We found several composite phenotypes which are more consistent within samples than between samples

Rich, heavily developed platform

Available Tools

Tools built for table-like data data structures and much better adapted to it. - K-Means, Correlation, PCA - Matrix Factorization, Genomics, Graph Analytics, Machine Learning

Commercial Support

Dozens of major companies (Apple, Google, Facebook, Cisco, …) donate over $30M a year to development of Spark and the Berkeley Data Analytics Stack - 2 startups in the last 6 months with seed-funding in excess of $15M each

Academic Support

- All source code is available on GitHub
- Elegant (20,000 lines vs my PhD of 75,000+)
- No patents or restrictions on usage
- Machine Learning Course in D-INFK next semester based on Spark

Beyond: Streaming

Post-processing goals

- Analysis done in weeks instead of months
- Some real-time analysis and statistics

Streaming

Can handle static data or live data coming in from a ‘streaming’ device like a camera to do real-time analysis. The exact same code can be used for real-time analysis and static code

Scalability

Connect more computers.

Start workers on these computer.

Beyond: Approximate Results

Projects at AMPLab like Spark and BlinkDB are moving towards approximate results. - Instead of `mean(volume)` - `mean(volume).within_time(5)` - `mean(volume).within_ci(0.95)`

For real-time image processing it might be the only feasible solution and could drastically reduce the amount of time spent on analysis.