Map-Reduce for Machine Learning on Multi-Core

Gary Bradski, work with Stanford:
Prof. Andrew Ng
Cheng-Tao, Chu, Yi-An Lin, Yuan Yuan Yu.
Sang Kyun Kim with Prof. Kunle Olukotun.
Ben Sapp, Intern

garybradski@gmail.com

Acknowledgments

The Map-Reduce work was supported by:

- Intel Corp*. and by
- DARPA ACIP program
- With much help from Stanford CS Dept.

- I left Intel a month ago to join Rexee Inc.

www.rexee.com

- I maintain my joint appointment as a consulting Professor at Stanford University CS Dept.

***Some slides taken from Intel presentation on Google's Map-Reduce and Paint slide from www.cs.virginia.edu/~evans/bio/slides/paintable.ppt**

And Ad-Sense Adverts

Open, Free for Commercial Use:

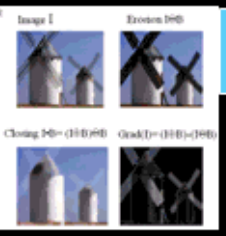
- Open Source Computer Vision Library
  - http://sourceforge.net/projects/opencvlibrary/
- Machine Learning Library
  - http://sourceforge.net/projects/opencvlibrary/

Optimized Support for Nominal $, Royalty free:

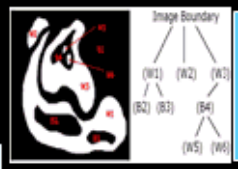
- Performance
  - http://www.intel.com/cd/software/products/asmo-na/eng/302910.htm

OpenCV: Open Source Computer Vision Lib.

http://www.intel.com/research/mrl/research/opencv



General Image Processing Functions



Geometric descriptors

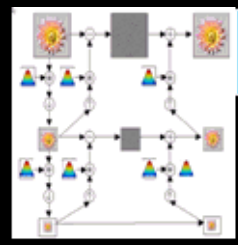
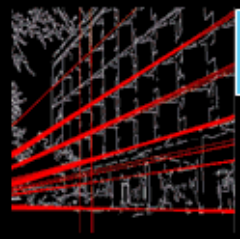
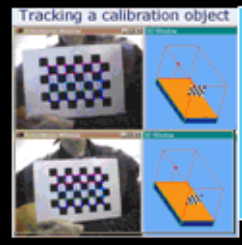


Image Pyramids

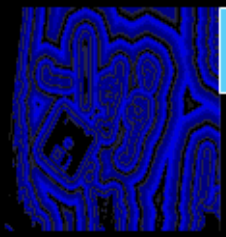
Segmentation



Features

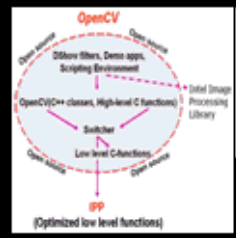


Camera calibration, Stereo, 3D

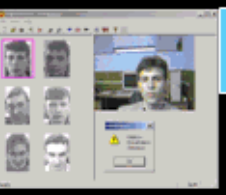


Measures

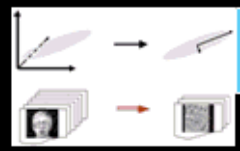
Tracking



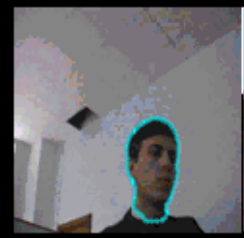
Utilities and Data Structures



Recognition



Matrix Math



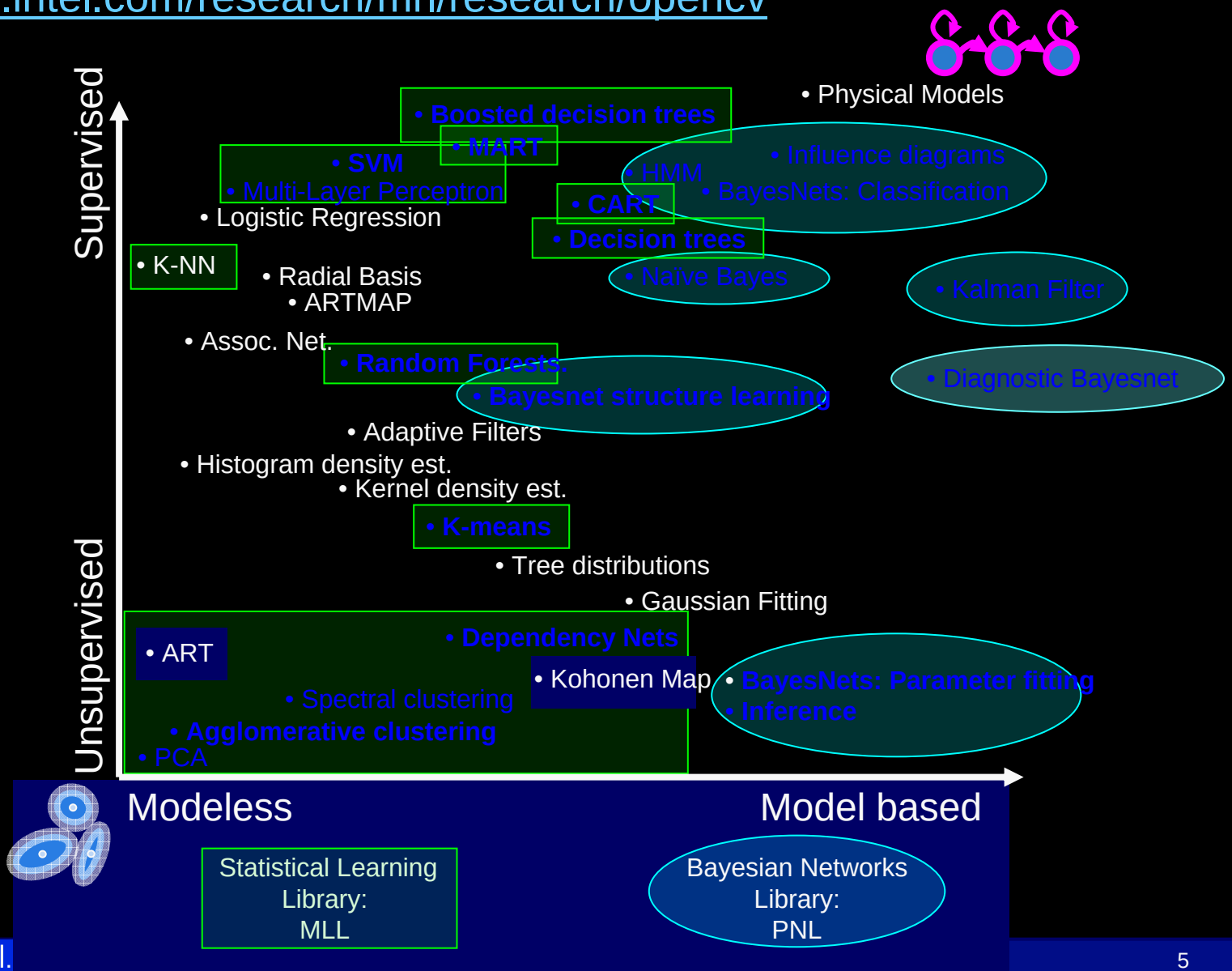
Fitting

- Takes advantage of Intel® Integrated Performance Primitives (optimized routines)

Machine Learning Libraries (MLL)

Subdirectory under OpenCV

http://www.intel.com/research/mrl/research/opencv



Key:

- **Optimized**
- Implemented
- Not implemented

Talk

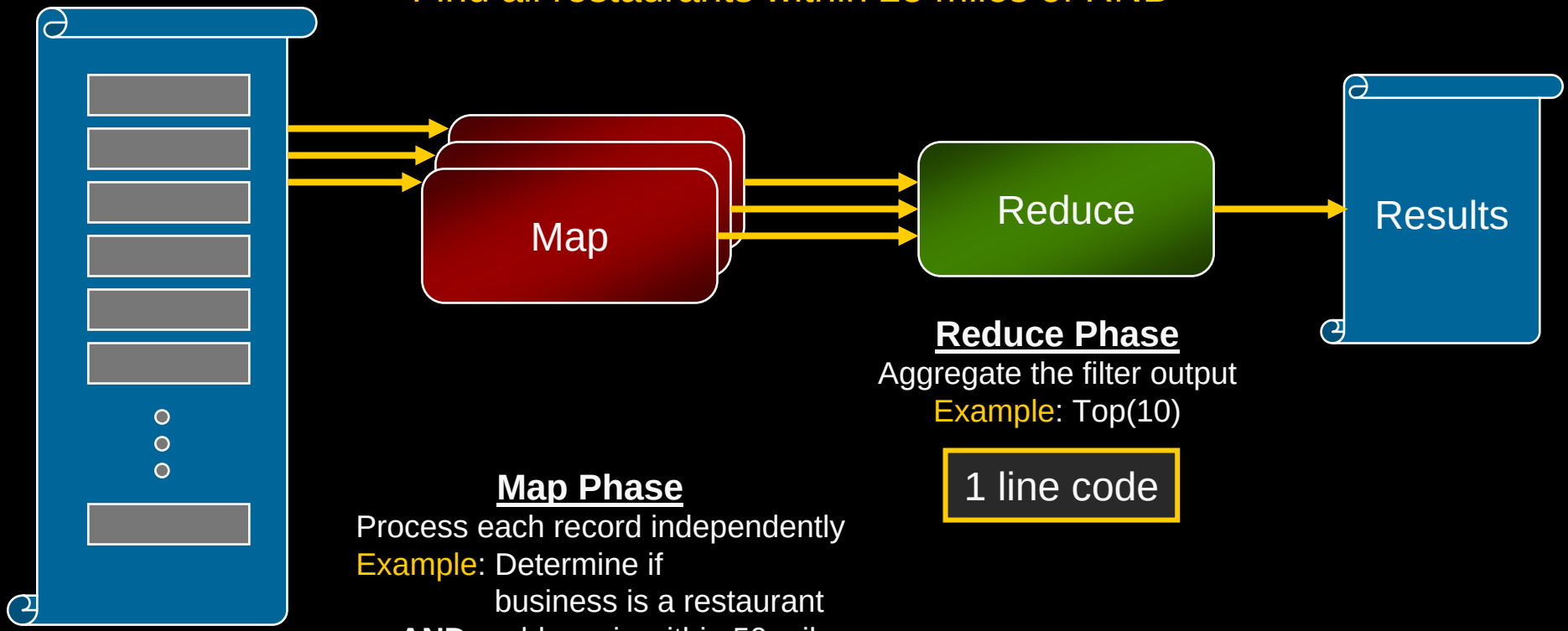
garybradski@gmail.com

Overview

- What is Map-Reduce?
- Can Map-Reduce be used on Multi-Core?
  - We developed a form for distributed algorithms:
  - These forms can be expressed in a Map-Reduce framework as a parallel API
  - Dynamic load balancing, Vision and MANY Core
- Current projects and speculations
  - Model enabled vision.

Google's Map-Reduce Programming Model

Find all restaurants within 25 miles of RNB



Input Data
A large collection of records
Stored across multiple disks
Example: Business Addresses

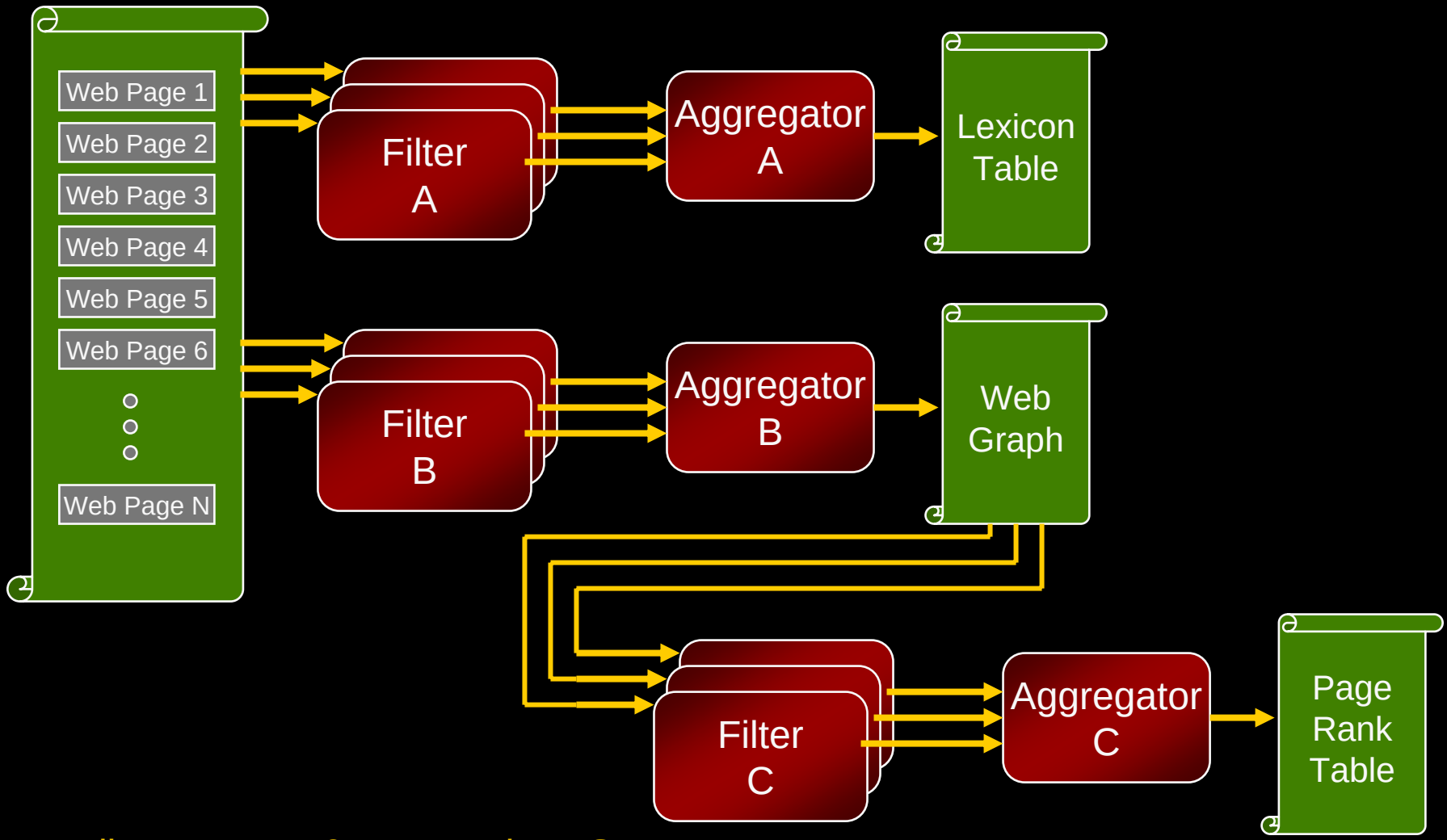
Map Phase
Process each record independently
Example: Determine if
business is a restaurant
AND address is within 50 miles

10s of lines code

Reduce Phase
Aggregate the filter output
Example: Top(10)

1 line code

More Detail:
Google Search Index Generation

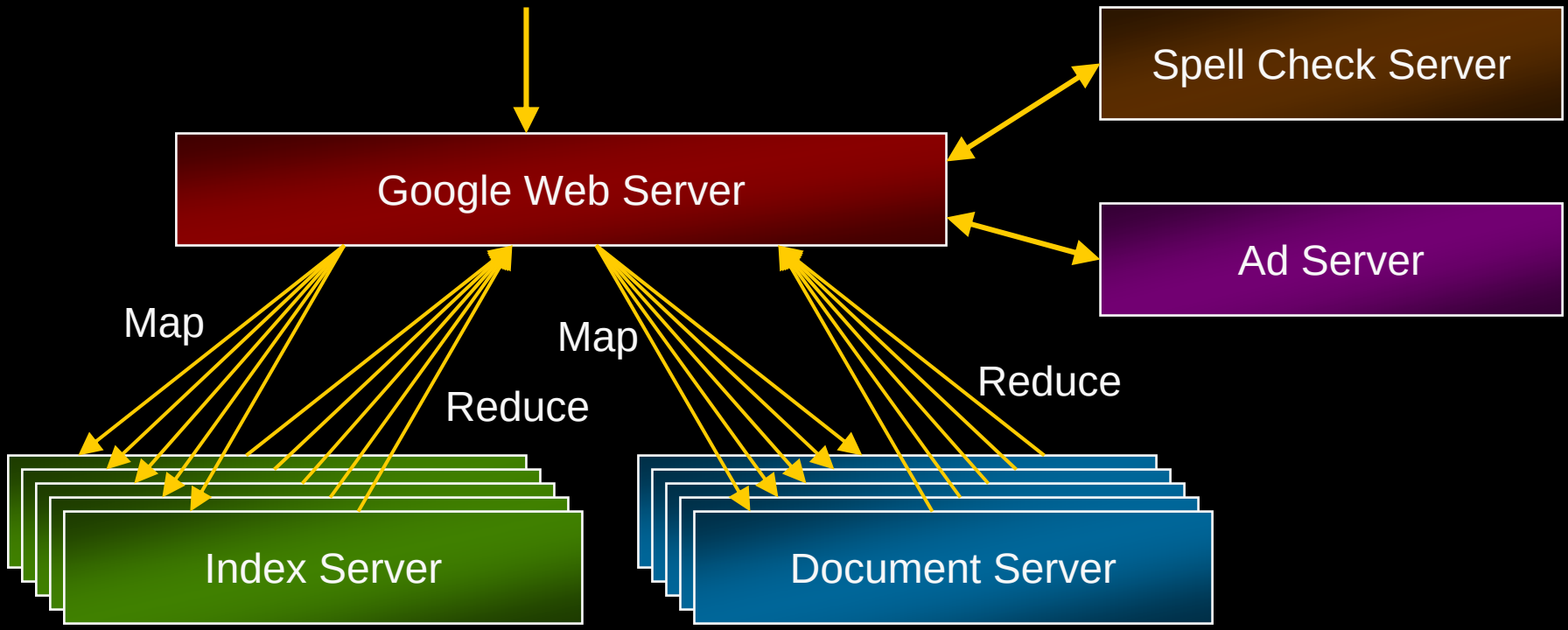


In Reality, Uses 5-10 Map-Reduce Stages

Efficiency impacts how often this is run. Impacts quality of the search.

More Detail: In Use:
Google Search Index Lookup

Processing a Google Search Query



Motivation

- Application: Sifting through large amounts of data
  Used for:
  - Generating the Google search index
  - Clustering problems for Google News and Froogle products
  - Extraction of data used to produce reports of popular queries
  - Large scale graph computation
  - Large scale machine learning
  - Generate reverse web link graph

- Platform: cluster of inexpensive machines
  - Scale to large clusters: Thousands of machines
  - Data distributed and replicated across machines of the cluster
  - Recover from disk and machine failures

Google Hardware Infrastructure

- Commodity PC-based Cluster
  - 15,000 Machines in 2003
  - Scalable and Cost Efficient

- Reliability in Software

  Use inexpensive disks (not RAID)

  Use commodity Server
  - Data Replication: Data survives crashes
  - Fault Tolerance: Detect and recover
    - Hardware Error
    - Bugs in Software
    - Misconfiguration of the Machine

Two Map-Reduce Programming Models

- **Sawzall**
  - Simpler interpreted language
  - Not expressive enough for everything (use of intermediate results).
- **MR-C++**
  - Library that allows full control over Map-Reduce
  - Can sometimes run much faster, but
  - Much slower to code/debug

Two Map-Reduce Programming Models

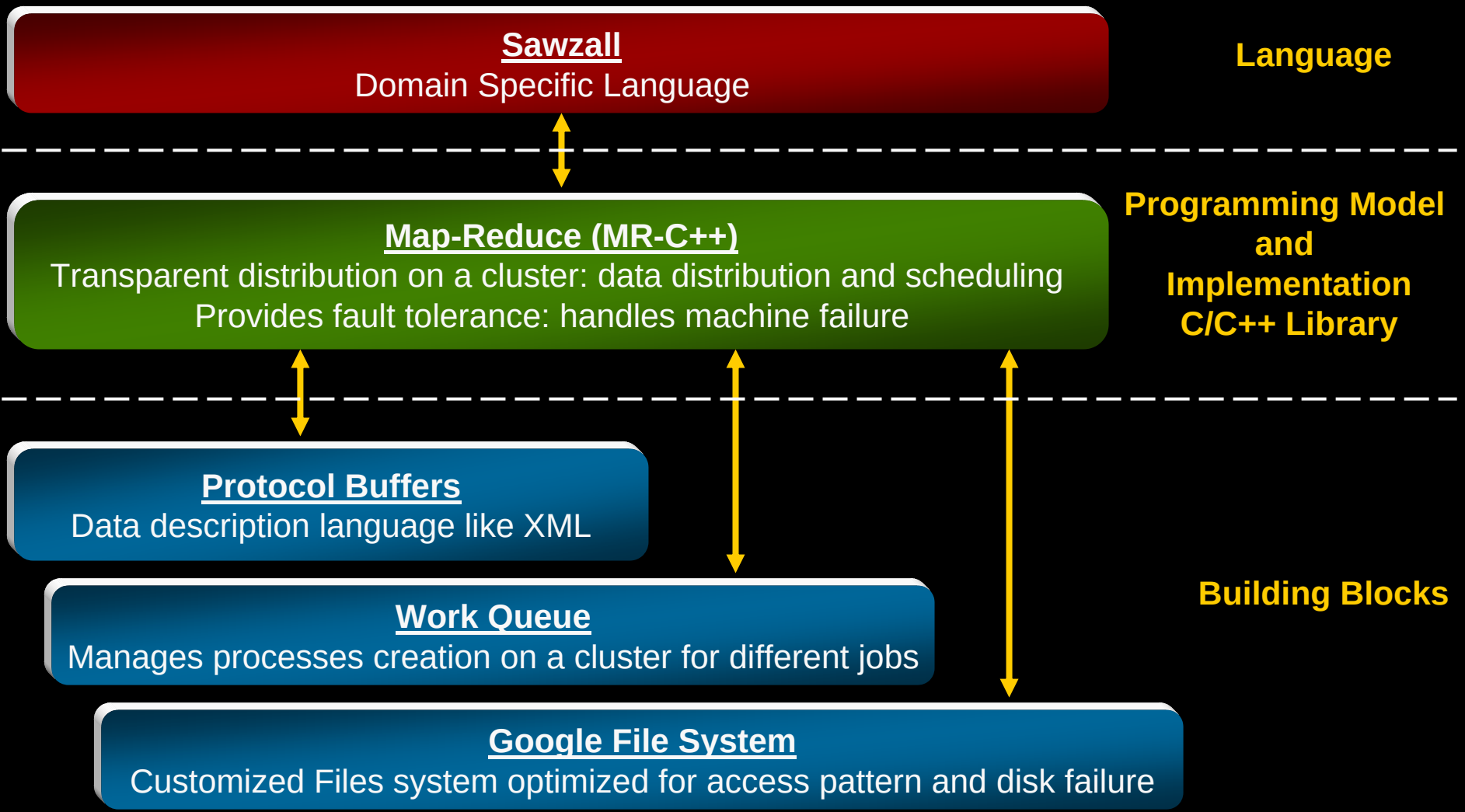
1. MR-C++

- C++ with Map-Reduce lib
  - Programmer writes Map and Reduce functions
- Relatively clean model
  - Allows workarounds
    - Side-effects, Combiner functions, Counters
- Simplifies programming
  - An order of magnitude less code than before
- Efficient (C++)

2. Sawzall

- Domain specific language
  - Programmer provides Map
  - Use built-in set of Reduce
- Clean model
  - Enforces the clean model
    - If it doesn't fit the model, use the Map-Reduce
- Even simpler programming
  - Often an order of magnitude less than Map-Reduce
- Inefficient + "Good enough"
  - Interpreter (50x slower)
  - Pure value semantics

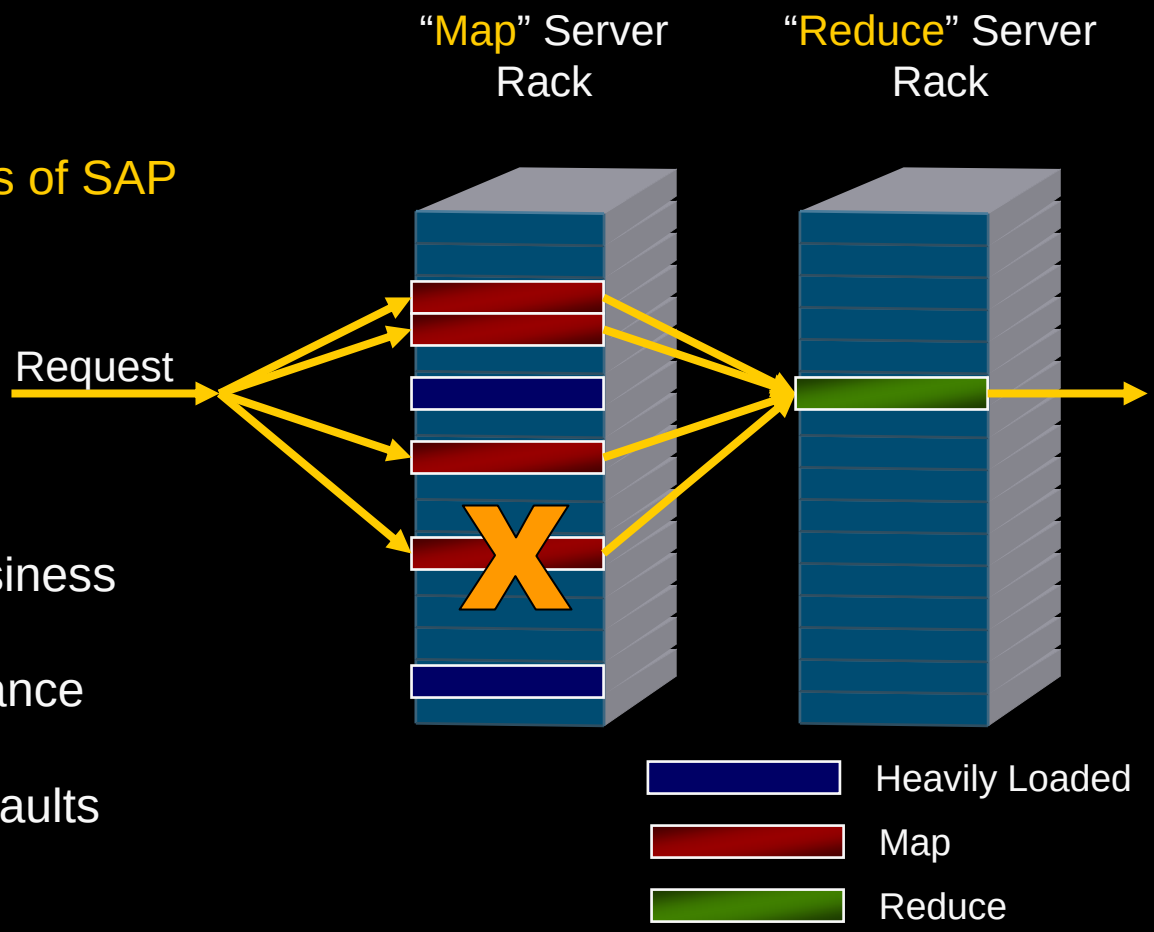
Map-Reduce Infrastructure Overview



Map-Reduce on a Cluster

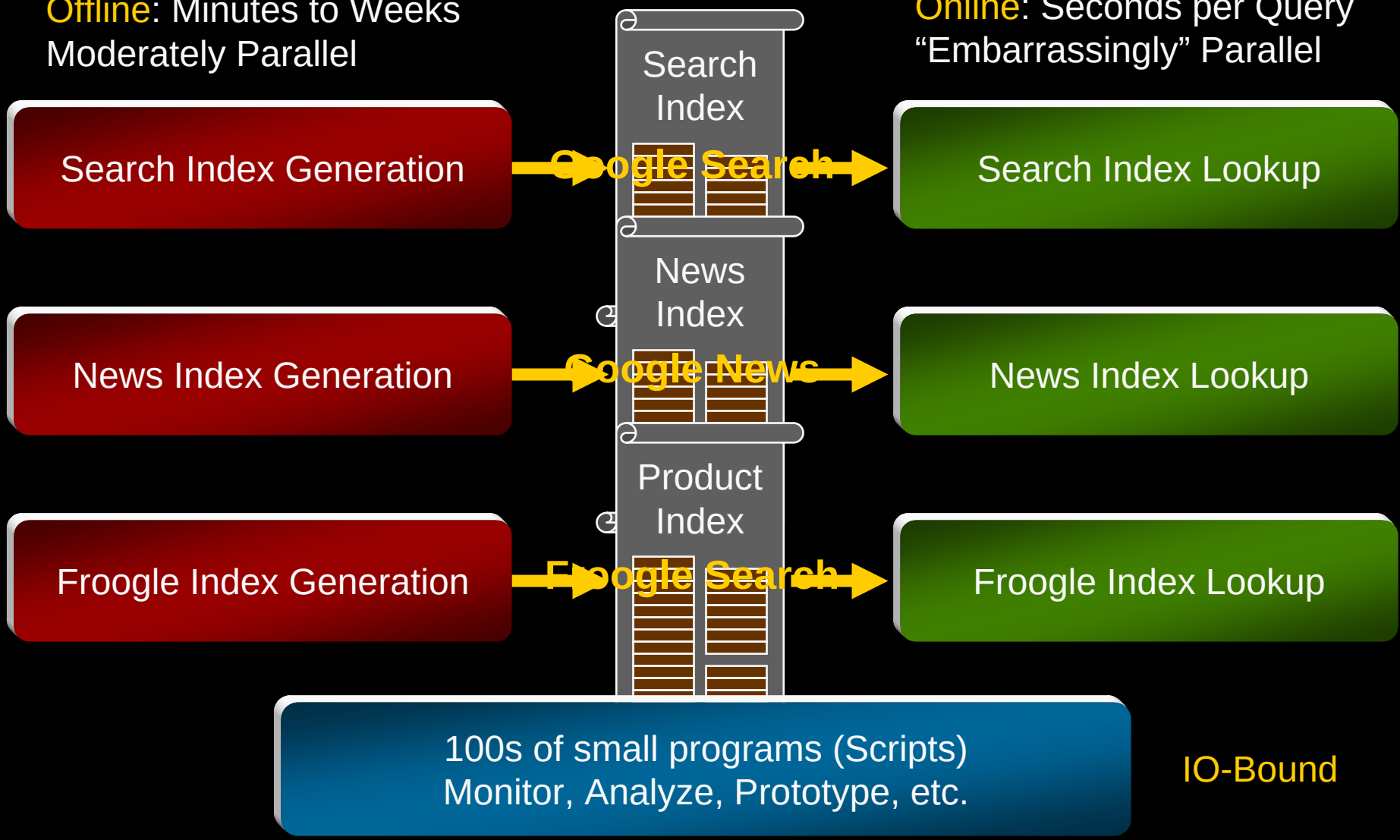
Find all restaurants within 25 miles of SAP

1. Use Cluster
   A. Locate Data
   B. Exploit Parallelism
2. Map Phase: Check each business
3. Collect the list
4. Reduce Phase: Sort by Distance
5. Report the results
6. At any stage: Recover from faults



These programs are complicated but similar.
Map-Reduce hides parallelism and distribution from programmer (steps 1, 3, 5, & 6).

Examples of Google Applications



Google on Many-Core Architecture

- Google Workload Characteristics
  - Extremely Parallel, Shared Working-Set
  - Throughput-oriented
  - Low Available ILP

  Confirmed by Google's IndexBench Benchmark

Extension to Mulit- and Many-Core?

- Hyperthreading → Multi-Core → Many-Core
  - Delivers more "Queries per Second per Watt"
    - Simpler cores (efficiently use transistors)
    - Reduce communication across machines
    - Benefit from sharing caches and Memory

Map-Reduce on Multi-Core

Use machine learning to study.

garybradski@gmail.com

Overview

- We developed a form for distributed algorithms:
  - Summation form
    - Done for Machine Learning and Computer vision
- These forms can be expressed in a Map-Reduce framework as a parallel API
- Map-Reduce
  - Lends itself to dynamic load balancing
- Papers in accepted in NIPS 2006, HPCA 2007:

Summation form for Machine Learning

- Basically, any algorithm fits that learns by:
  - Computes sufficient statistics
  - Updates by local gradients
  - Forms ~independent “voting” nodes

Machine Learning Summation Form:
Map Reduce Example

Simple example: locally weighted linear regression (LWLR):[1]

$$\theta^* = \min_\theta \sum_{i=1}^{m} w_i (\theta^T x_i - y_i)^2$$

To parallelize this algorithm, re-write it in "summation form" as follows:

$$A = \sum_{i=1}^{m} w_i (x_i x_i^T); b = \sum_{i=1}^{m} w_i (x_i y_i)$$

$$\theta^* = A^{-1} b$$

MAP: By writing LWLR as a summation over data points, we now easily parallelize the computation of *A* and *b* by dividing the data over the number of threads. For example, with two cores:

$$A_1 = \sum_{i=1}^{m/2} w_i (x_i x_i^T); A_2 = \sum_{i=m/2+1}^{m} w_i (x_i x_i^T)$$

REDUCE: A is obtained by adding A_1 and A_2 (at negligible additional cost). The computation for b is similarly parallelized.

[1] Here, the w_i are "weights." If all weights = 1, this reduces to the ordinary least squares.

ML fit with Many-Core

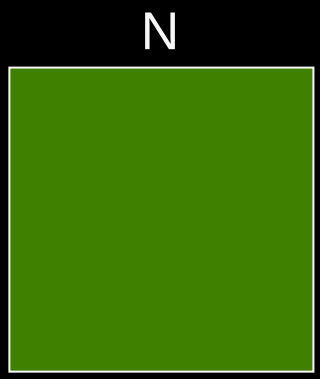
To be more clear:

We have a set of *M* data points each of length *N*

$$x_1 = x_1^1, x_1^2, \ldots, x_1^N$$
$$x_2 = x_2^1, x_2^2, \ldots, x_2^N$$
$$\ldots$$
$$x_M = x_M^1, x_M^2, \ldots, x_M^N$$

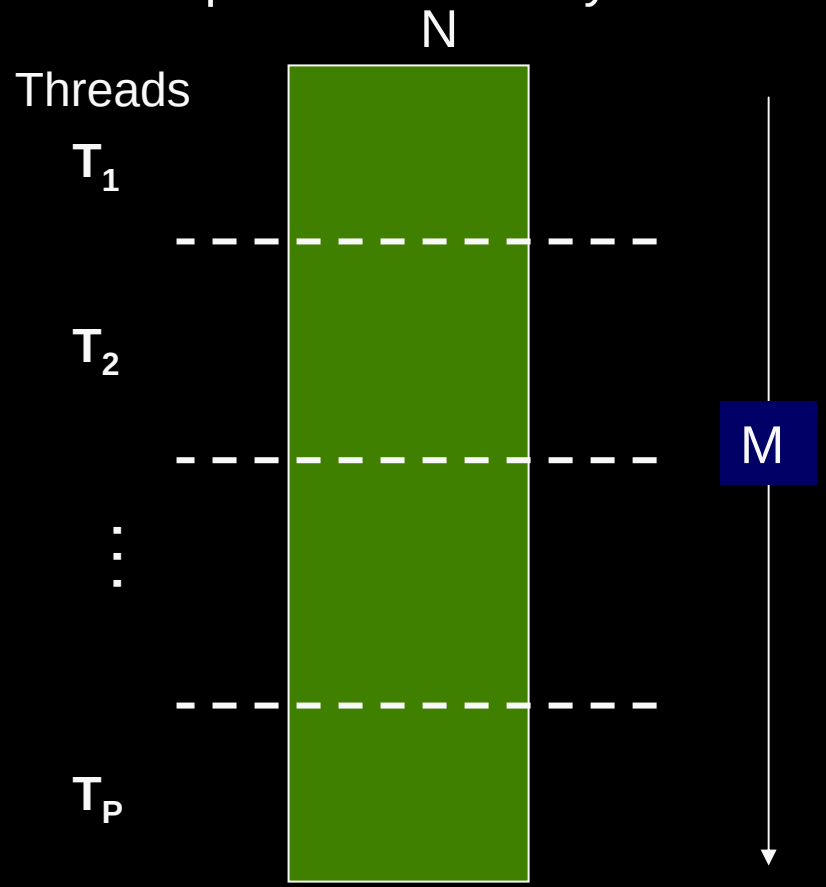
$$A = \sum_{i=1}^{m} w_i (x_i x_i^T); b = \sum_{i=1}^{m} w_i (x_i y_i)$$
$$\theta^* = A^{-1} b$$

Each $(x_i x_i^T)$ implies an NxN communication



N<<M or we haven't formulated a ML problem

Decomposition for Many-Core:



Each "**T**" incurres a NxN communication cost

This is a good fit for Map-Reduce

Map Reduce

We take as our starting point the functional programming construct “Map-Reduce”* and it’s use in Google allowing common massively parallel programming.

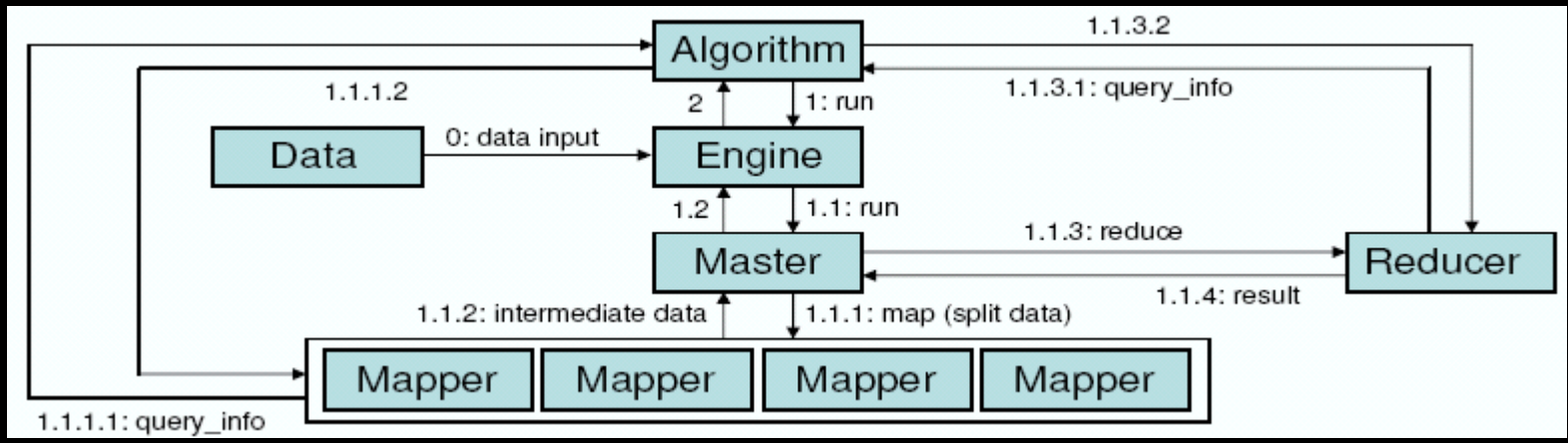
- Map-Reduce is a divide and conquer approach:
  - Map step: Processes subsets of the data in parallel with no communication between subsets.
    - Processing must result in “sparse” results such as sufficient statistics, small histograms, votes etc.
  - Reduce step: Aggregate (sum, tally, sort) the results of the map steps back to a central master.
  - There may be many map-reduce steps in a single algorithm
- We extend this concept to machine learning for multi-core
  - And will also extend to Computer vision next.

(*) Jeffrey Dean and Sanjay Ghemawat. “Mapreduce: Simplified data processing on large clusters”, *Operating Systems Design and Implementation*, pages 137–149, 2004.

Map-Reduce Machine Learning Framework

Programming Environment Block Diagram

- 0 Engine subdivides the data
- 1 Algorithm is run, appropriate engine invoked
  - 1.1 Master is invoked by engine
  - 1.1.1 Data is mapped out to cores for processing
  - 1.1.2 Sparse processed data from cores is returned
  - 1.1.3 And aggregated by reducer
  - 1.1.4 Results are returned
  - Other needed global info can be retrieved by query (1.1.1.1, 1.1.3.1)



List of Implemented and Analyzed Algorithms

	Algorithm	Have summation form?
1.	Linear regression	Yes
2.	Locally weighted linear regression	Yes
3.	Logistic regression	Yes
4.	Gaussian discriminant analysis	Yes
5.	Naïve Bayes	Yes
6.	SVM (without kernel)	Yes
7.	K-means clustering	Yes
8.	EM for mixture distributions	Yes
9.	Neural networks	Yes
10.	PCA (Principal components analysis)	Yes
11.	ICA (Independent components analysis)	Yes
12.	Policy search (PEGASUS)	Yes
13.	Boosting	Unknown
14.	SVM (with kernel)	Unknown
15.	Gaussian process regression	Unknown
16.	Fisher Linear Discriminant	Yes
17.	Multiple Discriminant Analysis	Yes
18.	Histogram Intersection	Yes

Analyzed:

Complexity Analysis:

m: number of data points; ***n***: number of features; ***P***: number of mappers (processors/cores); ***c***: iterations or nodes. Assume matrix inverse is O(n^3/P') where P' are iterations that may be parallelized.
For **L**ocally **W**eighted **L**inear **R**egression (**LWLR**) training: (**'** => transpose below):
==========
X'WX: mn^2
Reducer: $\log(P)n^2$
X'WY: mn
Reducer: $\log(P)n$
$(X'WX)^{-1}$: n^3/P'
$(X'WX)^{-1}X"WY$: n^2
Serial: $O(mn^2 + n^3/P')$
Parallel: $O(mn^2/P + n^3/P' + \log(P)n^2$
map invert reduce

Algorithms	single	multi
LWLR	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2\log(P))$
LR	$O(mn^2)$	$O(\frac{mn^2}{P} + n\log(P))$
NB	$O(mn + nc)$	$O(\frac{mn}{P} + nc\log(P))$
NN (3 layers)	$O(mn + nc)$	$O(\frac{mn}{P} + nc\log(P))$
GDA	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2\log(P))$
PCA	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2\log(P))$
ICA	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2\log(P))$
K Means	$O(mnc)$	$O(\frac{mnc}{P} + mn\log(P))$
EM	$O(mn^2 + n^3)$	$O(\frac{mn^2}{P} + \frac{n^3}{P'} + n^2\log(P))$
SVM	$O(mn)$	$O(\frac{mn}{P} + n\log(P))$

Basically: Linear speedup with increasing numbers of cores.

Some Results, 2 Cores

Dataset characteristics:

Data Sets	samples (m)	features (n)
Adult	30162	14
Helicopter Control	44170	21
Corel Image Features	68040	32
IPUMS Census	88443	61
Synthetic Time Series	100001	10
Census Income	199523	40
ACIP Sensor	229564	8
KDD Cup 99	494021	41
Forest Cover Type	581012	55
1990 US Census	2458285	68

Basically: Linear speed up with the number of cores

On 2 processor machine:

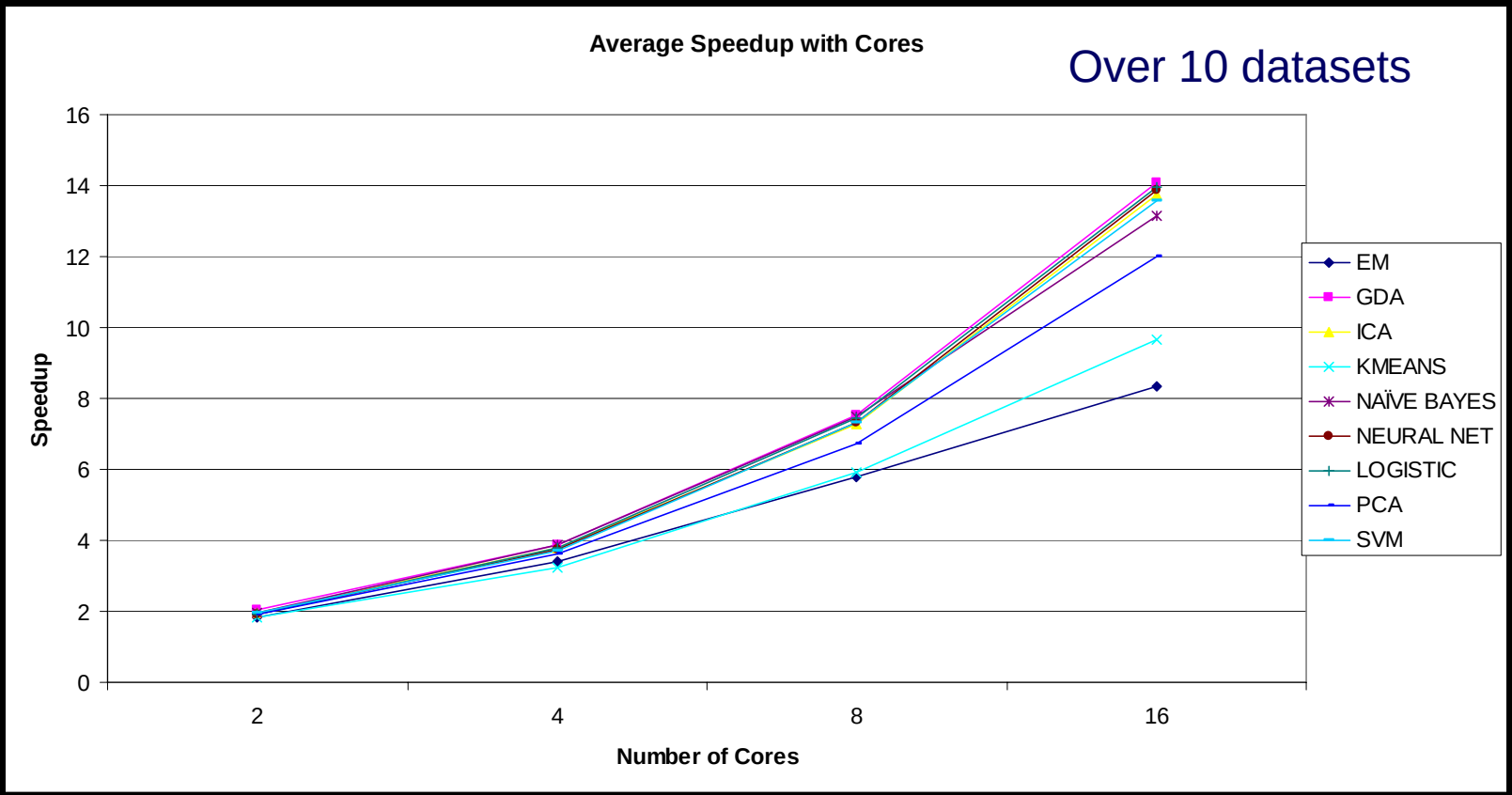
	lwlr	gda	nb	logistic	pca	ica	svm	nn	kmeans	em
Adult	1.922	1.801	1.844	1.962	1.809	1.857	1.043	1.825	1.947	1.854
Helicopter	1.93	2.155	1.924	1.92	1.791	1.856	1.435	1.847	1.857	1.86
Corel Image	1.96	1.876	2.002	1.929	1.97	1.936	1.427	2.018	1.921	1.832
IPUMS	1.963	2.23	1.965	1.938	1.965	2.025	1.655	1.974	1.957	1.984
Synthetic	1.909	1.964	1.972	1.92	1.842	1.907	1.403	1.902	1.888	1.804
Census Income	1.975	2.179	1.967	1.941	2.019	1.941	1.619	1.896	1.961	1.99
Sensor	1.927	1.853	2.01	1.913	1.955	1.893	1.468	1.914	1.953	1.949
KDD	1.969	2.216	1.848	1.927	2.012	1.998	1.593	1.899	1.973	1.979
Cover Type	1.961	2.232	1.951	1.935	2.007	2.029	1.636	1.887	1.963	1.991
Census	2.327	2.292	2.008	1.906	1.997	2.001	1.701	1.883	1.946	1.977
avg.	1.985	2.080	1.950	1.930	1.937	1.944	1.498	1.905	1.937	1.922

Results on 1-16 Way System

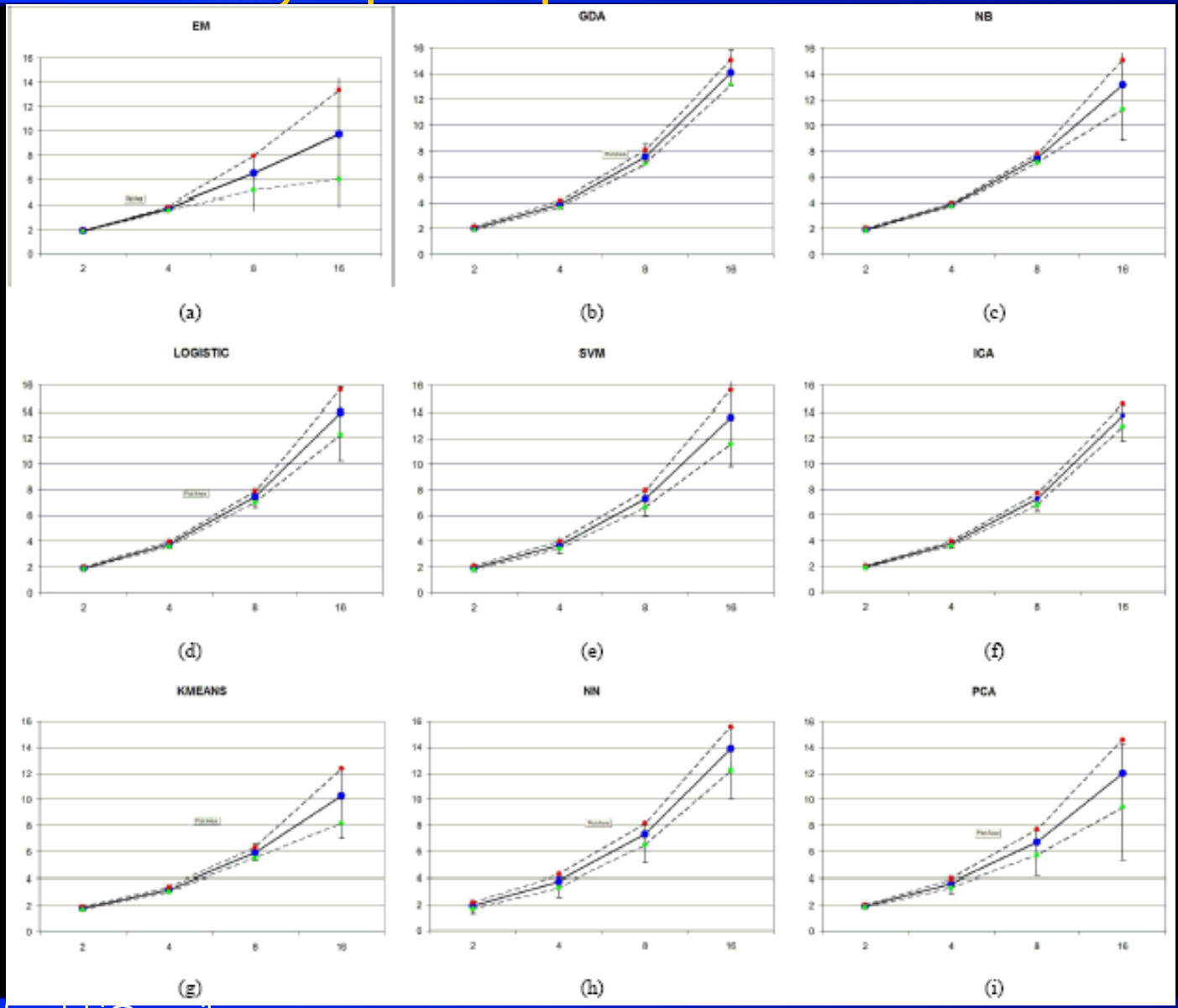
- In general, near linear speedup.
- Must have enough data to justify.
- Load time is often the bottleneck.

Average Speedup over All Data

- In general, near linear speedup.
- Must have enough data to justify.
- Load time is often the bottleneck.



Map-Reduce for Machine Learning:
Summary Speedups over 10 datasets



Bold line is avg.
Dashed is std.
Bars are max/min

RESULT:
Basically linear or slightly better speedup.

Overall Results

- **Speedup**
  - The speedup for the total time becomes dominated by load time.
  - The speedup for the workloads was near linear.
  - The speedups for some of the workloads were limited by the serial region.
    - Ex. PCA, EM, ICA, NR_LOGICSTIC
  - The speedups for one of the workloads was limited by the reducer.
    - Ex. K-mean
- **Workload vs. Data Set Size**
  - The workload time was roughly linear to the data set size, when the size was reasonably large enough.
- **Serial Time**
  - Most of the serial regions had only a fixed amount of computation.
  - The only exception is the PCA algorithm.
  - We can see that the impact of the serial regions lessons while we increase the data set size.
- **Reducer Execution Time**
  - The reducer time for most of the algorithms do not scale with the size of data.
    - Exception is the K-means algorithm, which increases linearly.
  - The reducer time, however, scales with the number of processors.
  - For most of the algorithms, the reducer execution time was insignificant.

Dynamic Run Time Optimization Opportunity

- In the Summation form framework
  - The granularity of the threads is flexible
  - May be changed globally over time

- Dynamic Optimization/Load Ballance Algorithm:
  - Keep subdividing threads
  - Until performance registers indicate performance fall-off.
  - Step back towards larger granularity
  - Periodically iterate

Discussion

garybradski@gmail.com

Really want a Macro/micro Map-Reduce

- For independent processing
  - Seemless over clusters of Multi-Core
- For 2-tier, learning across independent records
  - Map over clusters
  - Call Map-Reduce machine learning routines
- Should work well for algorithms like
  - Kleinberg's Stochastic Discrimination
  - Breiman's Random Forests

Extension to Other Areas of AI

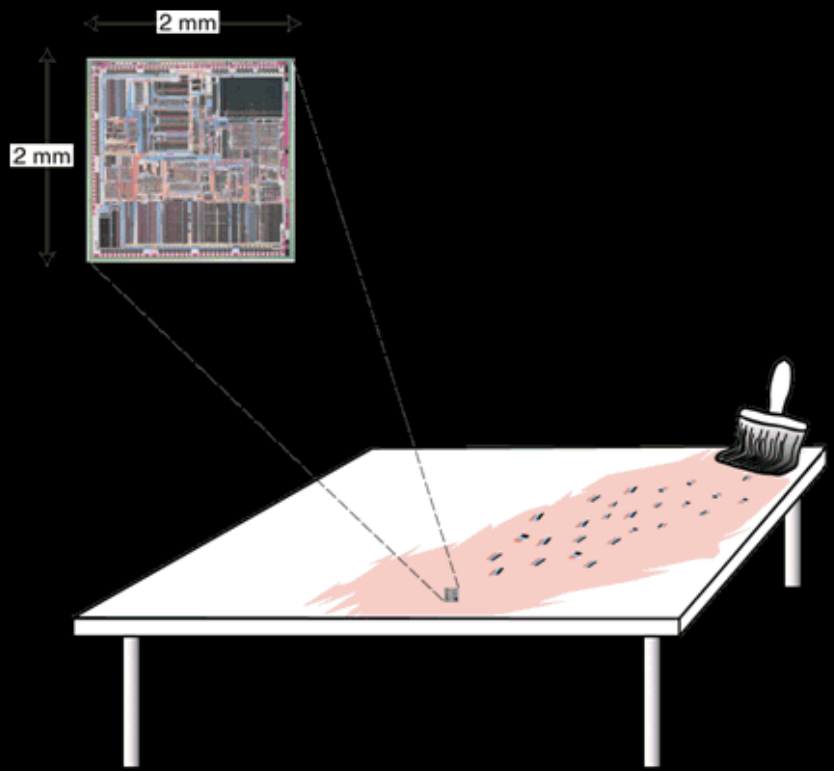
- We've seen machine learning
- Map Reduce on Multi-Core seems to extend
  - To computer vision
  - Navigation
  - Inference (loopy belief propagation)

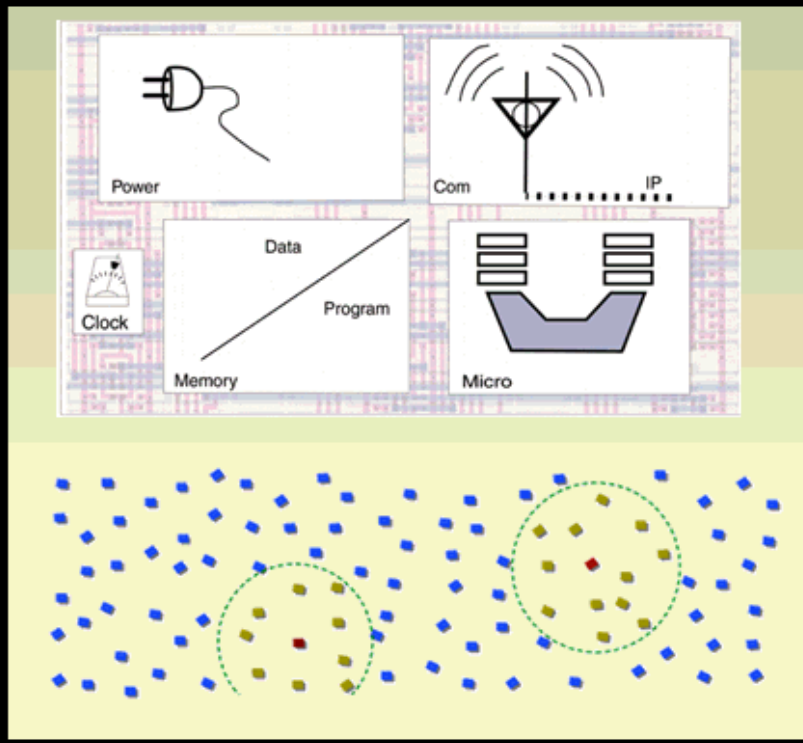
Multi-Core here => Shared Memory, but… Map-Reduce spans bridge to non-shared memory

- Multi-core will soon scale to a point where:
  - Shared memory doesn't work
  - Individual cores become unreliable
- Why not go all the way? Millions of cores.
- Map-Reduce variants should still work.

- **Paintable computers**. William Butera
  - MIT Media Lab Thesis
  - More extensions under MIT Professor Neil Gershonfeld's Center of Bits and Atoms.
  - Bill now calls it "Scale Free Architecture" SFA

Paintable Computing





Conclusions

- Map-Reduce is an effective paradigm for Google.
- It can be usefully extended to support AI algorithms on Multi-core
- By dropping explicit masters, it can span Scale Free Architectures with Millions of distributed nodes.

References and Related Work

garybradski@gmail.com

References

Map Reduce on Multi-core:

- Map-Reduce for Machine Learning on Multicore
  Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Ng, Kunle Olukotun
  Neural Information Processing Systems 2006.

- Evaluating MapReduce for Multi-core and Multiprocessor Systems
  Arun Penmetsa, Colby Ranger, Ramanan, Christos Kozyrakis and Gary Bradski.
  High-Performance Computer Architecture-13, 2007.

Map Reduce at Google:

- Interpreting the Data: Parallel Analysis with Sawzall
  Rob Pike, Sean Dorward, Robert Griesemer, and Sean Quinlan
  Scientific Programming Journal. Special Issue on Grids and Worldwide Computing Programming Models and Infrastructure.

- MapReduce: Simplified Data Processing on Large Clusters
  Jeffrey Dean and Sanjay Ghemawat
  Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

- The Google File System
  Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
  Symposium on Operating Systems Principles, Lake George, NY, October, 2003.

- Web Search for a Planet: The Google Cluster Architecture
  Luiz Barroso, Jeffrey Dean, and Urs Hoelzle
  In IEEE Micro, Vol. 23, No. 2, pages 22-28, March, 2003

Related Work

- Languages
  - Parallel Languages
    - Data Parallel Languages [date back to 1970s]
    - Functional Languages [at least 1960s]
    - Streaming Languages [date back to 1960s]
  - SQL
    - Map (“select-where”) and Reduce (sort, count, sum)
    - Multiple tables at a time (Join Operation)
    - Allows updates to tables

Related Work Cont'd

- Distributed Systems
  - Job scheduling [Condor, Active Disks, River]
  - Distributed sort [NOW-Sort]
  - Distributed, fault tolerant storage [AFS, xFS, Swift, RAID, NASD, Harp, Lustre]

Some Current Work, just for fun…

Statistical models of vision using
Fast multi-grid techniques from physics

garybradski@gmail.com

Questions

garybradski@gmail.com

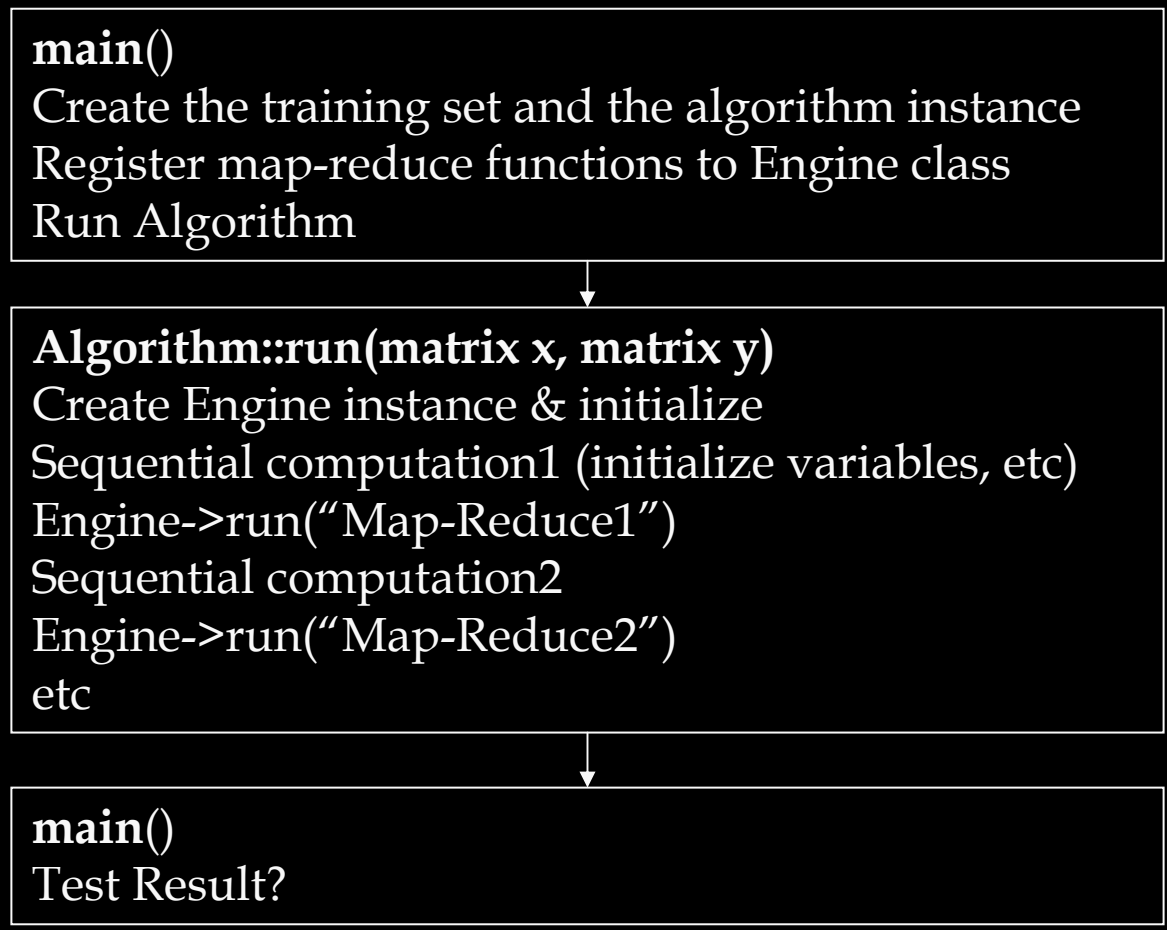
BACK UP

garybradski@gmail.com

Directory Structure of Code

- <algorithm>.cpp
  - These files are the multi-threaded version source codes of the machine learning algorithms.
- <algorithm>-single.cpp
  - These files are for single thread only, and is the same as the multi-threaded version except it does not have the overhead of using multiple threads.
- engine.cpp
  - The engine is responsible for creating multiple threads and return the computation result.
  - All threads must complete before returning.
- master.cpp, mapper.cpp, reduce.cpp
  - The functions in these files implement the map-reduce interaction.
- util.cpp
  - Reads input file into a matrix structure.
  - Records the time and has some debugging functions.
  - Other miscellaneous functions

Basic Flow For All Algorithms



Basic Flow For All Algorithms

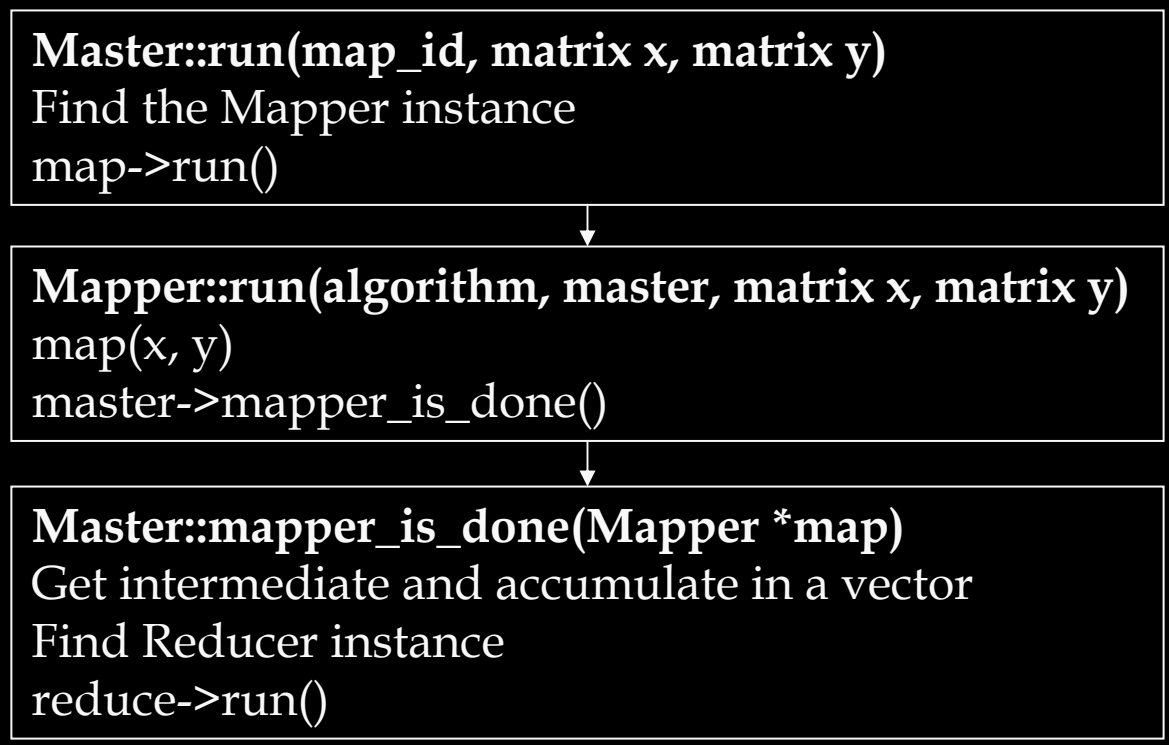
- What does the Engine do?

Engine::run(algorithm)
Create master instance
Create n threads (each execute Master::run())
Join n threads
Return master's result

  - All it does is create multiple threads and wait for the result.

Basic Flow For All Algorithms

- How does the Master work?



Basic Flow For All Algorithms

- How does the Master work? (cont')

Master::mapper_is_done(Mapper *map)
Get intermediate and accumulate in a vector
Find Reducer instance
reduce->run()

↓

Reducer::run(algorithm, master, matrix x, matrix y)
reduce(intermediate_data)
master->reducer_is_done()

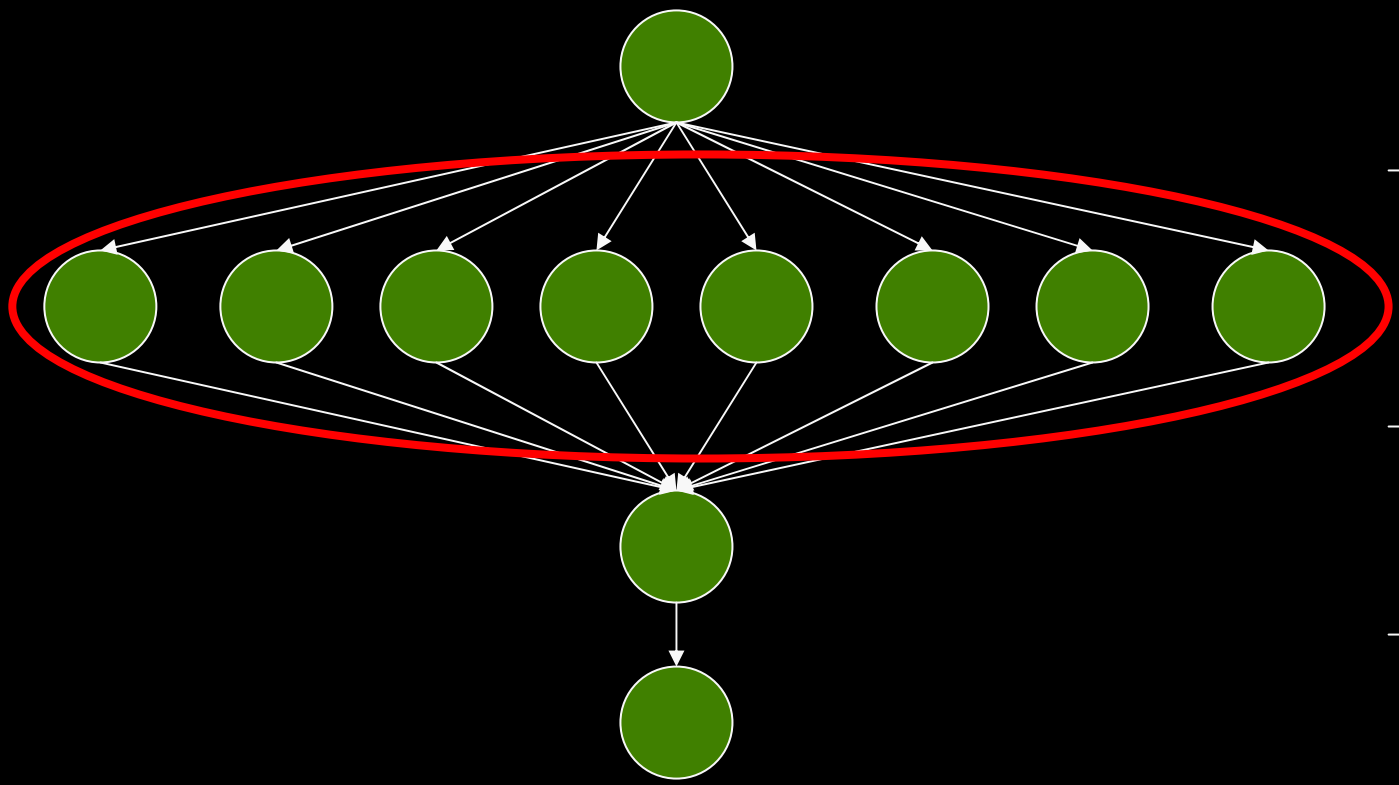
↓

Mapper::reducer_is_done()

  - Then Engine returns the result from master

Master::get_result()
reduce->get_result()

Where does parallelism takes place



Sequential Code

MAP

REDUCE

Sequential Code

Input Data

- A matrix of features
  - Each row is one input vector.
  - Each column corresponds to a separate feature.
  - indicates the i-th row vector

- Splitting the input matrix
  - The input matrix is split into n submatrix, where n is the number of threads.
  - Computation is are on input vectors; only 1-D decomposition is done.

ALGORITHM DESCRIPTION

EM Algorithm

- Expectation-Maximization Algorithm
  - Finds maximum likelihood estimates of parameters in probabilistic models.
  - Alternates between expectation (E) step and maximization (M) step.
- E step

- M step

EM Algorithm Map-Reduce

- x, w0, w1
  - x is the input data matrix where columns are features
  - w are vectors of probabilities of each row of features to map to a 0 or 1
  - Initially, w0 and w1 are some random values, satisfying $w0_j + w1_j = 1$
- class MStepMapper_PHI, class MStepReducer_PHI
  - The mapper sums $w0_j$ of its portion to produce intermediate data.
  - The reducer takes the partial sums and produces a global sum.
- class MStepMapper_MU, class MStepReducer_MU
  - The mapper obtains mu0 and mu1 by summing $w0_j * x_j$ and $w1_j * x_j$ respectively.
  - The reducer takes the partial mu0s and mu1s, and produces a global average mu0, mu1.
- class MStepMapper_SIGMA, class MStepReducer_SIGMA
  - The mapper produces a covariance matrix for each row and sums the covariance matrix to produce a partial sigma.
  - The reducer takes the partial sigma and produces a global sigma.
- class EStepMapper
  - Estimates w0 and w1 using the current phi, mu, and sigma values

EM Algorithm Performance Analysis

- Sequential code – possible parallelization
  - The reduce part of the code is executed by only one thread.
  - Reduction can be done in parallel, but this requires change in the current map-reduce structure of the code.
- Sequential code – inevitable
  - There is a part of code where sequential execution is inevitable.
  - Here, matrix LU decomposition is performed to obtain the inverse matrix and determinant of Sigma.
  - This part does not scale with the number of input, but only to the square of the number of features.
  - The maximum number of features with the current sensor data is 120, so it might be better to leave it sequential.
- Run on the sequential version of EM
  - Experimented with 1000 rows, 120 features on a Pentium-D machine

EM Algorithm Performance Analysis

- Run the single thread version of EM
  - Experimented with 100 and 1000 rows, 120 features on a Pentium-D
  - It was really EStepMapper that took up most of the execution time.
  - The reduction part was barely measureable at all.
  - The LU decomposition and inverse matrix part took a fixed amount of time(~14 sec) and did not change with the number of rows.

Logistic Algorithm

- Logistic Algorithm
  - Uses the logistic function, also called sigmoid function, as the hypotheses.

  - stochastic gradient ascent rule
    - Θ is the vector to be fitted. x is the feature matrix.
    - y is the target vector
    - gradient is defined as

    - stocastic gradient ascent rule

Logistic Algorithm code

- Label
  - The label is the target vector which has the same number of rows as the feature matrix.
  - The label was never initialized in this code, and most of any other codes.
  - This can lead to incorrect answers.
  - This shouldn't, however affect the runtime, and the input feature matrix itself isn't "valid" data anyways.
- Algorithm main loop
  - Loops until tolerance condition is met, or until the number of iteration reaches 10.
  - The tolerance condition is considered to be met when the norm of the gradient is less than 3.
  - The step size for the gradient is given as 0.0001
- class LogisticMapper
  - The mapper sums the gradient value for each row to get a partial sum of gradients.
  - The reducer sums the intermediate gradient values to produce a global gradient.

Logistic Algorithm code

- Sequential part
  - After getting the global gradient, it is scaled, added to the current theta, and is used to compute the norm.
  - All three operations are independent to the number of input data.
  - Thus this part consumes about fixed amount of time, it is not easy to parallelize, and it might not be worth the effort.

GDA Algorithm

- Gaussian Discriminant Analysis model
  - Essentially finding the maximum likelihood estimates of the parameters.

  - Almost same as the M step in the EM algorithm.

GDA Algorithm code

- class GDAMapper_PHI, class GDAReducer_PHI
  - The mapper sums the labels of its portion.
  - The reducer takes the partial sums and produces a global sum.
- class GDAMapper_MU, class GDAReducer_MU
  - The mapper obtains mu0 and mu1 by summing $\{label_j = 0\}*x_j$ and $\{label_j = 1\}*x_j$ respectively.
  - The reducer takes the partial mu0s and mu1s, and produces a global average mu0, mu1.
- class GDAMapper_SIGMA, class GDAReducer_SIGMA
  - The mapper produces a covariance matrix for each row and sums the covariance matrix to produce a partial sigma.
  - The reducer takes the partial sigma and produces a global sigma.
- Sequential code
  - Similar as to the EM algorithm, the size of the sequential part is fixed to the number of columns (i.e. features).

K-means Algorithm

- K-means Algorithm
  - A variant of the EM algorithm.
  - Clusters objects based on attributes into k partitions.
  - The objective is to minimize total intra-cluster variance.

  - Here, μ is the centroid, or mean point of all the points

K-means Algorithm code

- centroid
  - Initially some dummy value (the first k rows of the feature matrix)
  - Gets updated while running the algorithm in multiple iterations
- class KMeansMapper
  - The mapper places each row to a cluster by its similarity to the centroid.
  - The similarity is measured by the dot-product between the input row vector and the centroid vector.
  - The feature row vector is sent to the cluster of which centroid showed the maximum similarity.
- class KMeansReducer
  - The reducer accumulates each centroid with input vectors in the same cluster.
  - The centroid is scaled down by the number of elements in the cluster, making the centroid the mean of the input vectors.
  - This part of the code is executed by only one thread.
  - However, the reduce runtime is only 1% of the map runtime.
  - Parallelizing this part would need either a lock, or private centroids with additional reduction, which means overhead.

NB Algorithm

- Naive Bayes Algorithm
  - Based on the assumption that x_i's condition is independent of y.
  - Discretizes the data in this implementation into 8 categories.
  - To make a prediction, just compute

NB Algorithm source code

- Discretization
  - Before running the algorithm, the input is discretized into 8 categories per column (i.e. feature).
- class NB_Mapper
  - The mapper counts the number of occurrence per category per column, once given y=0 and once given y=1.
  - Stores the counted data for each category in a 8x2 matrix called "table" for that column.
- class NB_Reducer
  - Gets the intermediate data and sums them into one 8x2 matrix.
  - Stores the "probability" per category for each column.

PCA Algorithm

- Principle Component Analysis
  - Only requires eigenvector computations.
  - The problem is essentially maximizing the following value with a unit length u:
  - This gives the principle eigenvector of is the covariance matrix.

PCA Algorithm code

- class PCAPrepMeanMapper, class PCAPrepMeanReducer
  - The mapper adds the row vectors to find the mean vector of its portion of data.
  - The reducer takes the partial mean and produces a global mean.
- class PCAPrepVar1Mapper, class PCAPrepVar1Reducer
  - The mapper obtains the variance vector of its portion of data.
  - The reducer takes the partial variance vectors and sum them together into a global variance vector.
- Dividing the input data by the variance vector
  - This is done by the main thread sequentially, although there is a mapper and reducer for this that was never used.
- class PCAMapper, class PCAReducer
  - The mapper produces a covariance matrix for each row and sums the covariance matrix to produce a partial covariance matrix.
  - The reducer takes the partial covariance matrix and produces a global covariance matrix.
- Sequential code
  - Computes the eigenvector of the covariance matrix

ICA Algorithm

- Independent Component Analysis
  - A computational method for separating a multivariate signal into additive subcomponents supposing the mutual statistical independence of the non-Gaussian source.
  - This is the case where there is a source vector that is generated from independent sources, and our goal is to recover the sources.
  - Let . Our goal is to find W so that we can recover the sources.
  - W is obtained and updated by iteratively computing the following rule:
  - g is the sigmoid function and alpha is the learning rate.

ICA Algorithm code

- class ICAMapper, class ICAReducer
  - The mapper computes the part of the gradient and sums them to create a partial gradient.
  - The reducer takes the partial gradient and sums them to a global gradient.
- Computing the invert matrix of W
  - This part is done in serial.
  - W is only dependent on the number of features, so the computation time of this part is fixed to the square of the number of features.
  - The main thread performs the LU decomposition and computes the transpose invert matrix of W.
  - This is added to the gradient and the gradient is scaled by the learning rate.
- W is computed and updated for 10 iterations.

LWLR Algorithm

- LWLR Algorithm
  - Might stand for “locally weighted linear regression”, but I’m not sure.
  - Get A matrix and b vector, then solve the linear equation to get theta.
  - The weight vector for this code is all 1s, which does not matter since it does not affect the performance.

LWLR Algorithm code

- class LWLR_A_Mapper, class LWLR_A_Reducer
  - The mapper computes the partial A matrix for its portion of data.
  - The reducer sums the partial A matrix to create a global A matrix.
- class LWLR_B_Mapper, class LWLR_B_Reducer
  - The mapper computes the partial b vector for its portion of data.
  - The reducer sums the partial b vector to create a global b vector.
- Householder solver for linear algebra
  - This part is serial, executed by the main thread.
  - The main thread uses householder solver to solve the linear equation.
  - Both the size of A and b is independent of the number of inputs. Therefore the computation time for this part is fixed.