Data Intensive Scalable Computing

Finding the Right Programming Models

Randal E. Bryant
Carnegie Mellon University

http://www.cs.cmu.edu/~bryant
Outline

Data-Intensive Scalable Computing
- Focus on petabytes, not petaflops
- Rethinking machine design

Map/Reduce Programming Model
- Suitability for wide class of computing tasks

Strengths & Limitations
- Scalability
- Performance

Beyond Map/Reduce
- Small variations
- Other programming models
Our Data-Driven World

Science
■ Data bases from astronomy, genomics, natural languages, seismic modeling, …

Humanities
■ Scanned books, historic documents, …

Commerce
■ Corporate sales, stock market transactions, census, airline traffic, …

Entertainment
■ Internet images, Hollywood movies, MP3 files, …

Medicine
■ MRI & CT scans, patient records, …
Why So Much Data?

We Can Get It
- Automation + Internet

We Can Keep It
- Seagate Barracuda
- 1.5 TB @ $76
  5¢ / GB
  (vs. 40¢ in 2007)

We Can Use It
- Scientific computation
- Business applications
- Harvesting Internet data

Deal of the Day

Seagate Barracuda 1.5 TB Desktop Hard Drive
Seagate Barracuda drives deliver reliable service and high performance. The Barracuda family has been developed and refined over 12 generations and embodies the highest levels of design for reliability and performance. While Barracuda LP is the low power leader, the entire family of Barracuda drives lead with... read more

List Price: $199.99
Yesterday's Price: $94.99
Today's Discount: -$19.00

Gold Box Price: $75.99 (62% off)

11 Comments | ★★★★★ (337)
## Oceans of Data, Skinny Pipes

No more blaming connection speeds for your losses.

Verizon FiOS – the fastest Internet available.

### Easy to store

- 1 Terabyte

### Hard to move

<table>
<thead>
<tr>
<th>Disks</th>
<th>MB / s</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seagate Barracuda</td>
<td>115</td>
<td>2.3 hours</td>
</tr>
<tr>
<td>Seagate Cheetah</td>
<td>125</td>
<td>2.2 hours</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Networks</th>
<th>MB / s</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home Internet</td>
<td>&lt; 0.625</td>
<td>&gt; 18.5 days</td>
</tr>
<tr>
<td>Gigabit Ethernet</td>
<td>&lt; 125</td>
<td>&gt; 2.2 hours</td>
</tr>
<tr>
<td>PSC Teragrid Connection</td>
<td>&lt; 3,750</td>
<td>&gt; 4.4 minutes</td>
</tr>
</tbody>
</table>
Data-Intensive System Challenge

For Computation That Accesses 1 TB in 5 minutes

- Data distributed over 100+ disks
  - Assuming uniform data partitioning
- Compute using 100+ processors
- Connected by gigabit Ethernet (or equivalent)

System Requirements

- Lots of disks
- Lots of processors
- Located in close proximity
  - Within reach of fast, local-area network
Google Data Centers

Dalles, Oregon
- Hydroelectric power @ 2¢ / KW Hr
- 50 Megawatts
  - Enough to power 6,000 homes
- Engineered for maximum modularity & power efficiency
- Container: 1160 servers, 250KW
- Server: 2 disks, 2 processors
High-Performance Distributed Computing: Two Versions

**Grid Computing**

Connect small number of big machines

- Allow resource sharing among supercomputers

**Issues**

- Programs must usually be specialized for machine
- Hard to get cooperation from multiple organizations

**DISC System**

Build big system from many small machines

- Use distributed systems principles to construct large-scale machine
- File system provides distribution, reliability, recovery
- Dynamically scheduled task as basic processing unit
Programming Model Comparison

**Bulk Synchronous**
- Commonly used for compute-intensive applications

**Map/Reduce**
- Commonly used for data-intensive applications

**Issues**
- Raw performance
- Scalability
  - Cost required to increase machine size by $K$
    - $\geq K$
  - Performance achieved by increasing machine size by $K$
    - $\leq K$
  - Frequency and impact of failures
**Bulk Synchronous Programming**

**Solving Problem Over Grid**
- E.g., finite-element computation

**Partition into Regions**
- p regions for p processors

**Map Region per Processor**
- Local computation sequential
- Periodically communicate boundary values with neighbors
**Typical HPC Operation**

**Characteristics**
- Long-lived processes
- Make use of spatial locality
- Hold all program data in memory (no disk access)
- High bandwidth communication

**Strengths**
- High utilization of resources
- Effective for many scientific applications

**Weaknesses**
- Requires careful tuning of application to resources
- Intolerant of any variability
Map/Reduce Programming Model

- Map computation across many objects
  - E.g., $10^{10}$ Internet web pages
- Aggregate results in many different ways
- System deals with issues of resource allocation & reliability

Dean & Ghemawat: “MapReduce: Simplified Data Processing on Large Clusters”, OSDI 2004
Create an word index of set of documents

- Map: generate \( \langle \text{word}, \text{count} \rangle \) pairs for all words in document
- Reduce: sum word counts across documents
Map/Reduce Operation

Characteristics

- Computation broken into many, short-lived tasks
  - Mapping, reducing
- Use disk storage to hold intermediate results

Strengths

- Great flexibility in placement, scheduling, and load balancing
- Can access large data sets

Weaknesses

- Higher overhead
- Lower raw performance
System Comparison: Programming Models

Conventional HPC

- Programs described at very low level
  - Specify detailed control of processing & communications
- Rely on small number of software packages
  - Written by specialists
  - Limits classes of problems & solution methods

DISC

- Application programs written in terms of high-level operations on data
- Runtime system controls scheduling, load balancing, …
HPC Fault Tolerance

Checkpoint
- Periodically store state of all processes
- Significant I/O traffic

Restore
- When failure occurs
- Reset state to that of last checkpoint
- All intervening computation wasted

Performance Scaling
- Very sensitive to number of failing components
Map/Reduce Fault Tolerance

Data Integrity
- Store multiple copies of each file
- Including intermediate results of each Map / Reduce
  - Continuous checkpointing

Recovering from Failure
- Simply recompute lost result
  - Localized effect
- Dynamic scheduler keeps all processors busy
DISC Scalability Advantages

- Distributed system design principles lead to scalable design
- Dynamically scheduled tasks with state held in replicated files

Provisioning Advantages

- Can use consumer-grade components
  - maximizes cost-performance
- Can have heterogenous nodes
  - More efficient technology refresh

Operational Advantages

- Minimal staffing
- No downtime
Getting Started

Goal

- Get people involved in DISC

Software

- Hadoop Project
  - Open source project providing file system and Map/Reduce
  - Supported and used by Yahoo
  - Rapidly expanding user/developer base
  - Prototype on single machine, map onto cluster
Exploring Parallel Computation Models

DISC + Map/Reduce Provides Coarse-Grained Parallelism

- Computation done by independent processes
- File-based communication

Observations

- Relatively “natural” programming model
- Research issue to explore full potential and limits
**Example: Sparse Matrices with Map/Reduce**

Task: Compute product $C = A \cdot B$

Assume most matrix entries are 0

**Motivation**

- Core problem in scientific computing
- Challenging for parallel execution
- Demonstrate expressiveness of Map/Reduce

\[
A = \begin{bmatrix}
10 & 20 \\
30 & 40 \\
50 & 60 & 70 \\
\end{bmatrix}
\quad X = \begin{bmatrix}
-1 \\
-2 & -3 \\
-4 \\
\end{bmatrix}
\quad B = \begin{bmatrix}
-10 & -80 \\
-60 & -250 \\
-170 & -460 \\
\end{bmatrix}
\]

\[
C = A \cdot B = \begin{bmatrix}
-10 & -80 \\
-60 & -250 \\
-170 & -460 \\
\end{bmatrix}
\]
Computing Sparse Matrix Product

- Represent matrix as list of nonzero entries
  \(\langle \text{row, col, value, matrixID} \rangle\)

- Strategy
  - Phase 1: Compute all products \(a_{i,k} \cdot b_{k,j}\)
  - Phase 2: Sum products for each entry \(i,j\)
  - Each phase involves a Map/Reduce
Group values $a_{i,k}$ and $b_{k,j}$ according to key $k$
Phase 1 “Reduce” of Matrix Multiply

- Generate all products $a_{i,k} \cdot b_{k,j}$
Phase 2 Map of Matrix Multiply

- Group products $a_{i,k} \cdot b_{k,j}$ with matching values of $i$ and $j$

Key = row, col

Key = 1,1

1

-10

C

Key = 1,2

1

-80

C

Key = 2,1

2

-60

C

Key = 2,2

2

-90

C

Key = 3,1

3

-120

C

Key = 3,2

3

-280

C

-50

A

-180

C

-80

C

-160

C

-280

C
Phase 2 Reduce of Matrix Multiply

- **Key = 1,1**
  - $1 \frac{-10}{C} \rightarrow 1$

- **Key = 1,2**
  - $1 \frac{-80}{C} \rightarrow 2$

- **Key = 2,1**
  - $2 \frac{-60}{C} \rightarrow 1$

- **Key = 2,2**
  - $2 \frac{-90}{C} \rightarrow 2$
  - $2 \frac{-160}{C} \rightarrow 2$

- **Key = 3,1**
  - $3 \frac{-120}{C} \rightarrow 1$
  - $3 \frac{-50}{A} \rightarrow 1$

- **Key = 3,2**
  - $3 \frac{-280}{C} \rightarrow 2$
  - $3 \frac{-180}{C} \rightarrow 2$

- **C**
  - $\begin{bmatrix} -10 & -80 \\ -60 & -250 \\ -170 & -460 \end{bmatrix}$

- **Sum products to get final entries**
Lessons from Sparse Matrix Example

**Associative Matching is Powerful Communication Primitive**
- Intermediate step in Map/Reduce

**Similar Strategy Applies to Other Problems**
- Shortest path in graph
- Database join

**Many Performance Considerations**
- Kiefer, Volk, Lehner, TU Dresden
- Should do systematic comparison to other sparse matrix implementations
Tweaking MapReduce

Map/Reduce/Merge
- Yang, et al. Yahoo & UCLA
- Add merge step to do tagged merge of data sets
  - E.g., getting elements from matrices A & B
- Demonstrate expressive power of relational algebra

Local Iterations
- Kambatla, et al., Purdue
- Support local and global iterations
- Step toward bulk synchronous model
Local / Global Map/Reduce

Iterative Computations
- e.g., PageRank

Graph Partitioning
- Partition into clusters
- Colocate data for each cluster

Computation
- Compute solution for each cluster, holding inter-cluster values constant
- Update inter-cluster values
Pig Project

- Chris Olston, Yahoo!
- Part of Apache/Hadoop

Merge Database & Programming
- SQL-like query language
- Set-oriented function application

Implementation
- Map onto Hadoop
- Automatic selection / optimization of algorithm
- Captures low-level tricks programmers have devised for Map/Reduce
Generalizing Map/Reduce

- E.g., Microsoft Dryad Project

## Computational Model

- Acyclic graph of operators
  - But expressed as textual program
- Each takes collection of objects and produces objects
  - Purely functional model

## Implementation Concepts

- Objects stored in files or memory
- Any object may be lost; any operator may fail
- Replicate & recompute for fault tolerance
- Dynamic scheduling
  - # Operators >> # Processors
Implementation Challenges

Hadoop Platform is Blessing
■ Large community adding features, improving performance
■ Easy to deploy
■ Growing body of documentation and materials
■ Works well enough for range of applications

... And Curse
■ Map & reduce functionality hardwired
■ Not designed as extensible platform

Dryad Not Widely Adopted
■ 305 citations on Google Scholar vs. 1759 Map/Reduce
■ Built on .NET
Conclusions

Distributed Systems Concepts Lead to Scalable Machines

- Loosely coupled execution model
- Lowers cost of procurement & operation

Map/Reduce Gaining Widespread Use

- Hadoop makes it widely available
- Great for some applications, good enough for many others

Lots of Work to be Done

- Richer set of programming models and implementations
- Expanding range of applicability
  - Problems that are data and compute intensive
  - The future of supercomputing?