ApacheCon North America 2011

Scaling Hadoop Applications

Milind Bhandarkar, Greenplum, A Division of EMC
Milind.Bhandarkar@emc.com, November 10, 2011

Presented by
The Apache Software Foundation
http://www.apache.org/

Produced by
STONECIRCLE PRODUCTIONS, LLC

Outline

- Scalability of Applications
- Causes of Sublinear Scalability
- Best Practices
- Q & A

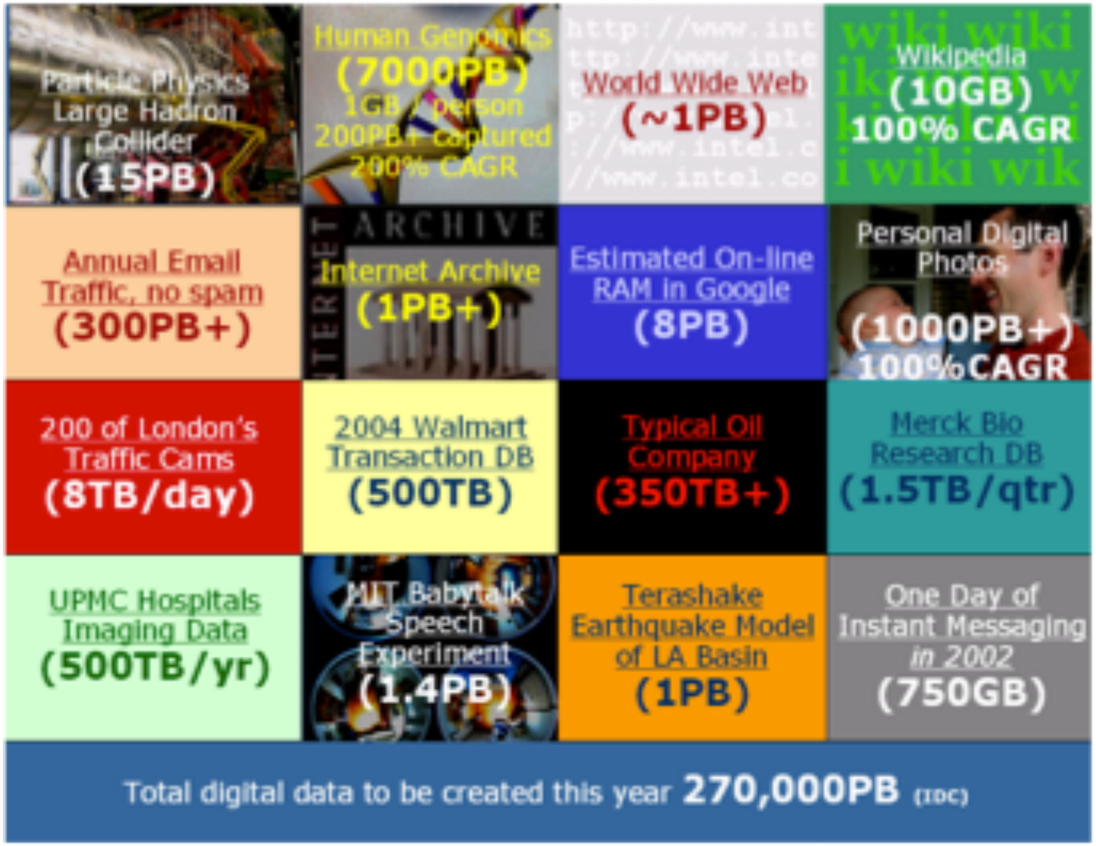
Who am I

- http://www.linkedin.com/in/milindb
- Chief Architect, Greenplum Labs, EMC
- Focused on Hadoop for 5+ years
  - Contributor since v 0.1
  - Founding member of Hadoop team at Yahoo
  - Built and led Grid solutions team at Yahoo!
  - Parallel programming for 20+ years

Importance of Scaling

- Zynga's Cityville: 0 to 100 Million users in 43 days ! (http://venturebeat.com/2011/01/14/zyngas-cityville-grows-to-100-million-users-in-43-days/)
- Facebook in 2009 : From 150 Million users to 350 Million ! (http://www.facebook.com/press/info.php?timeline)
- LinkedIn in 2010: Adding a member per second ! (http://money.cnn.com/2010/11/17/technology/linkedin_web2/index.htm)
- Public WWW size: 26M in 1998, 1B in 2000, 1T in 2008 (http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html)
- Scalability allows dealing with success gracefully !

Explosion of Data



(Data-Rich Computing theme proposal. J. Campbell, et al, 2007)

Apache Hadoop

- Simplifies building parallel applications
- Programmer specifies Record-level and Group-level sequential computation
- Framework handles partitioning, scheduling, dispatch, execution, communication, failure handling, monitoring, reporting etc.
- User provides hints to control parallel execution

Scalability of Parallel Programs

- If one node processes k MB/s, then N nodes should process (k*N) MB/s
- If some fixed amount of data is processed in T minutes on one node, the N nodes should process same data in (T/N) minutes
- Linear Scalability

Goal: Reduce Latency

http://www.flickr.com/photos/adhe55/2560649325/

Minimize program execution time

Goal: Increase Throughput

http://www.flickr.com/photos/mikebaird/3898801499/

Maximize data processed per unit time

Three Equations

- Amdahl's Law
- Little's Law
- Message Cost Model

Amdahl's Law

$$S = \frac{N}{1 + \alpha(N - 1)}$$

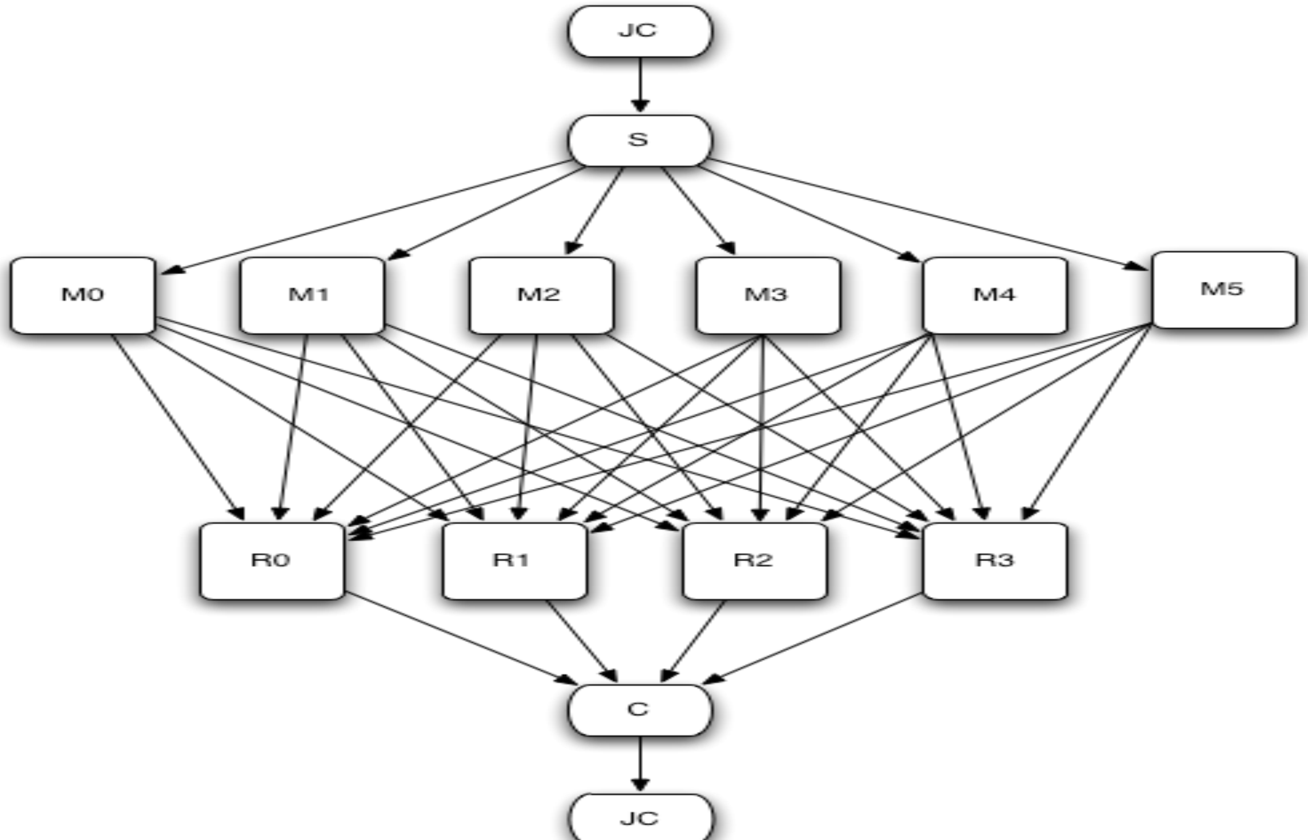
Multi-Phase Computations

- If computation C is split into N different parts, $C_1..C_N$
- If partial computation C_i can be speeded up by a factor of S_i

Amdahl's Law: Restated

$$S = \frac{\sum_{i=1}^{N} C_i}{\sum_{i=1}^{N} \frac{C_i}{S_i}}$$

MapReduce Workflow



Little’s Law

$$L = \lambda W$$

Message Cost Model

$$T = \alpha + N\beta$$

Message Granularity

- For Gigabit Ethernet
  - α = 300 μS
  - β = 100 MB/s
- 100 Messages of 10KB each = 40 ms
- 10 Messages of 100 KB each = 13 ms

Alpha-Beta

- Common Mistake: Assuming that α is constant
  - Scheduling latency for responder
  - Jobtracker/Tasktracker time slice inversely proportional to number of concurrent tasks
- Common Mistake: Assuming that β is constant
  - Network congestion
  - TCP incast

Causes of Sublinear Scalability

- Sequential Bottlenecks
- Load Imbalance
- Critical Paths
- Algorithmic Overheads
- Synchronization
- Granularity / Communication Overheads

Sequential Bottlenecks

- Mistake: Single reducer (default)
  - mapred.reduce.tasks
  - Parallel directive
- Mistake: Constant number of reducers independent of data size
  - default_parallel

Load Imbalance

- Unequal Partitioning of Work
- Imbalance = Max Partition Size – Min Partition Size
- Heterogeneous workers
  - Example: Disk Bandwidth
    - Empty Disk: 100 MB/s
    - 75% Full disk: 25 MB/s

Over-Partitioning

- For N workers, choose M partitions, M >> N
- Adaptive Scheduling, each worker chooses the next unscheduled partition
- Load Imbalance = $Max(Sum(W_k)) – Min(Sum(W_k))$
- For sufficiently large M, both Max and Min tend to $Average(W_k)$, thus reducing load imbalance

Critical Paths

- Maximum distance between start and end nodes
- Long tail in Map task executions
  - Enable Speculative Execution
- Overlap communication and computation
  - Asynchronous notifications
  - Example: deleting intermediate outputs after job completion

Algorithmic Overheads

- Repeating computation in place of adding communication
- Parallel exploration of solution space results in wastage of computations
- Need for a coordinator
- Share partial, unmaterialized results
  - Consider memory-based small replicated K-V stores with eventual consistency

Synchronization

- Shuffle stage in Hadoop, M*R transfers
- Slowest system components involved: Network, Disk
- Can you eliminate Reduce stage ?
- Use combiners ?
- Compress intermediate output ?
- Launch reduce tasks before all maps finish

Task Granularity

- Amortize task scheduling and launching overheads
- Centralized scheduler
  - Out-of-band Tasktracker heartbeat reduces latency, but
  - May overwhelm the scheduler
- Increase task granularity
  - Combine multiple small files
  - Maximize split sizes
  - Combine computations per datum

Hadoop Best Practices - 1

- Use higher-level languages (e.g. Pig)
- Coalesce small files into larger ones & use bigger HDFS block size
- Tune buffer sizes for tasks to avoid spill to disk
- Consume only one slot per task
- Use combiner if local aggregation reduces size

Hadoop Best Practices - 2

- Use compression everywhere
  – CPU-efficient for intermediate data
  – Disk-efficient for output data
- Use Distributed Cache to distribute small side-files (<< than input split size)
- Minimize number of NameNode/ JobTracker RPCs from tasks

Contact

- Milind Bhandarkar
  - Milind.Bhandarkar@emc.com
  - Twitter: @techmilind
  - http://www.tumblr.com/blog/milindb