Big Data: Scale Down, Scale Up, Scale Out

Phillip B. Gibbons

Intel Science & Technology Center for Cloud Computing

Keynote Talk at IPDPS’15
May 28, 2015
ISTC for Cloud Computing

$11.5M over 5 years + 4 Intel researchers. Launched Sept 2011

25 faculty
87 students
(CMU + Berkeley, GA Tech, Princeton, Washington)

Underlying Infrastructure enabling the future of cloud computing

www.istc-cc.cmu.edu
whenever the **volume** or **velocity** of data overpowers current processing systems/techniques, resulting in **performance** that falls far short of desired

This talk: Focus on performance as key challenge

Many other challenges, including:

- variety of data, veracity of data
- analytics algorithms that scale
- programming
- security, privacy
- insights from the data, visualization
Three approaches to improving performance by orders of magnitude are:

- **Scale down** the amount of data processed or the resources needed to perform the processing

- **Scale up** the computing resources on a node, via parallel processing & faster memory/storage

- **Scale out** the computing to distributed nodes in a cluster/cloud or at the edge
Scale down the amount of data processed or the resources needed to perform the processing.

Goal: Answer queries much faster/cheaper than brute force

- Specific query? memoized answer
- Family of queries?
  - Retrieval? good index
  - With underlying common subquery (table)? materialized view
- Aggregation? data cube

Important Scale Down tool: approximation (w/error guarantees)
• Scale Down Insight:
  Often **EXACT** answers not required
  - DSS applications usually *exploratory*: early feedback to help identify “interesting” regions
  - **Preview** answers while waiting. **Trial** queries
  - **Aggregate queries**: precision to “last decimal” not needed
Fast Approximate Answers

Often, only interested in leading digits of answer

E.g., Average salary for...

- $59,152.25 (exact) in 10 minutes
- $59,000 +/- $500 (with 95% confidence) in 10 seconds

Orders of magnitude speed-up because synopses are orders of magnitude smaller than original data
The Aqua Architecture

Data Warehouse

Network

SQL Query Q

HTML XML

Browser Excel

Result

Warehouse Data Updates

Big Data: Scale Down, Scale Up, Scale Out

© Phillip B. Gibbons
The Aqua Architecture

Picture with Aqua:

- Aqua is middleware, between client & warehouse (Client: + error bound reporting. Warehouse SW: unmodified)

- Aqua Synopses are stored in the warehouse

- Aqua intercepts the user query and rewrites it to be a query Q’ on the synopses. Data warehouse returns approximate answer
Precomputed, Streaming Synopses

Our Insights (circa 1996)

- Precomputed is often faster than on-the-fly
  - Better access pattern than sampling
  - Small synopses can reside in memory

- Compute synopses via one pass streaming
  - Seeing entire data is very helpful: provably & in practice (Biased sampling for group-bys, Distinct value sampling, Join sampling, Sketches & other statistical functions)
  - Incrementally update synopses as new data arrives

Bottom Line:
Orders of magnitude faster on DSS queries
Example: Distinct-Values Queries

```sql
select count(distinct target-attr) from rel where P
```

Template

```sql
select count(distinct o_custkey) from orders where o_orderdate >= '2014-05-28'
```

Example using TPC-D/H/R schema

- How many distinct customers placed orders in past year?
  - Orders table has many rows for each customer, but must only count each customer once & only if has an order in past year
Distinct-Values Query Approaches

• **Estimate from Random Sample**
  - Statistics, Databases, etc
  - Lousy in practice
  - [Charikar’00] Need linear sample size

• **Flajolet-Martin’85**
  - One-pass algorithm, stores $O(\log u)$ bits
  - Only produces count, can’t apply a predicate

• **Our Approach: Distinct Sampling** [VLDB’01]
  - One-pass, stores $O(t \times \log u)$ tuples
  - Yields sample of distinct values, with up to $t$-size uniform sample of rows for each value
  - First to provide provably good error guarantees
Over the entire range of skew:

- Distinct Sampling has 1.00-1.02 ratio error
- At least 25 times smaller relative error than GEE and AE
Scale Down Today

• Hundreds and hundreds of clever algorithms
  – Synopsis-based approximations tailored to query families
  – Reduce data size, data dimensionality, memory needed, etc

• Synopses routinely used in Big Data analytics applications at Google, Twitter, Facebook, etc
  – E.g., Twitter’s open source Summingbird toolkit
    • Hyperloglog – number of unique users who perform a certain action; followers-of-followers
    • CountMin Sketch – number of times each query issued to Twitter search in a span of time; building histograms
    • Bloom Filters – keep track of users who have been exposed to an event to avoid duplicate impressions
      (10^8 events/day for 10^8 users)

[Boykin et al, VLDB’14]
How to Tackle the Big Data Performance Challenge

• **Scale Down**

• **Scale Up** the computing resources on a node, via parallel processing & faster memory/storage

• **Scale Out**
Why **Scale Up** when you can **Scale Out**?

- **Much of Big Data focus has been on Scale Out**
  - Hadoop, etc

- **But if data fits in memory of multicore then often order of magnitude better performance**
  - GraphLab1 (multicore) is 1000x faster than Hadoop (cluster)
  - Multicores now have 1-12 TB memory: most graph analytics problems fit!

- **Even when data doesn’t fit, will still want to take advantage of Scale Up whenever you can**
Multicore: 144-core Xeon Haswell E7-v3

socket

- 2 HW threads
- 32KB
- 256KB
- 45MB Shared L3 Cache

... 18

socket

- 2 HW threads
- 32KB
- 256KB
- 45MB Shared L3 Cache

... 8

socket

- 2 HW threads
- 32KB
- 256KB
- 45MB Shared L3 Cache

... 18

up to 12 TB Main Memory

Attach: Hard Drives & Flash Devices
Hierarchy Trends

• Good performance [energy] requires effective use of hierarchy

• Hierarchy getting richer
  – More cores
  – More levels of cache
  – New memory/storage technologies
    • Flash/SSDs, emerging PCM
    • Bridge gaps in hierarchies – can’t just look at last level of hierarchy
Hi-Spade: Hierarchy-Savvy Sweet Spot

Platform 1

Hierarchy-Savvy

(Pain)-Fully Aware

Platform 2

Ignoring

programming effort

performance

Goals: Modest effort, good performance in practice, robust, strong theoretical foundation
What Yields Good Hierarchy Performance?

- **Spatial locality**: use what’s brought in
- **Temporal locality**: reuse it
- **Constructive sharing**: don’t step on others’ toes

Two design options
- **Cache-aware**: Focus on the bottleneck level
- **Cache-oblivious**: Design for any cache size

Stepping on toes
- e.g., all CPUs write B at ≈ the same time
**Multicore Hierarchies’ Key New Dimension: Scheduling**

Scheduling of parallel threads has **LARGE** impact on hierarchy performance.

Recall our problem scenario:
all CPUs want to write B at ≈ the same time.

Can mitigate (but not solve) if can **schedule** the writes to be far apart in time.
Program-centric Analysis

• Start with a portable program description: dynamic Directed Acyclic Graph (DAG)

• Analyze DAG without reference to cores, caches, connections...

Program-centric metrics

• Number of operations (Work, W)
• Length of Critical Path (Depth, D)
• Data reuse patterns (Locality)

Our Goal: Program-centric metrics + Smart thread scheduler delivering provably good performance on many platforms
Parallel Cache Complexity Model

Decompose task into maximal subtasks that fit in space $M$ & glue operations

Cache Complexity $Q^*(M,B) =$

$\sum$ Space for $M$-fitting subtasks

$+ \sum$ Cache miss for every access in glue

$M,B$ parameters either used in algorithm (cache-aware) or not (cache-oblivious)

[Simhadri, 2013]
Space-Bounded Scheduler
[Chowdhury, Silvestri, Blakeley, Ramachandran IPDPS’10]

Key Ideas:

• Assumes space use (working set sizes) of tasks are known (can be suitably estimated)

• Assigns a task to a cache C that fits the task’s working set. Reserves the space in C. Recurses on the subtasks, using the CPUs and caches that share C (below C in the diagram)

Cache costs: optimal $\sum_{\text{levels}} Q^*(M_i) \times C_i$ where $C_i$ is the miss cost for level $i$ caches

Experiments on 32-core Nehalem: reduces cache misses up to 65% vs. work-stealing
Sharing vs. Contention

Sharing: operations that share the same memory location (or possibly other resource)

Contention: serialized access to a resource (potential performance penalty of sharing)

Replace **concurrent update** with **Priority Update**: updates only if higher priority than current
Priority Update has Low Contention under High Sharing

Perform poorly under high sharing

Perform well under high sharing

5 runs of $10^8$ operations on 40-core Intel Nehalem
Further Research Directions

• **Determinism** at function call abstraction, Commutative Building Blocks, Deterministic Reservations for loops, Use of priority update [PPoPP’12, SPAA’13, SODA’15]

• **Scaling Up** by redesigning algorithms & data structures to take advantage of new storage/memory technologies [VLDB’08, SIGMOD’10, CIDR’11, SIGMOD’11, SPAA’15]
How to Tackle the Big Data Performance Challenge

• **Scale Down**

• **Scale Up**

• **Scale Out** the computing to distributed nodes in a cluster/cloud or at the edge
Big Learning Frameworks & Systems

- **Goal**: Easy-to-use programming framework for Big Data Analytics that delivers good performance on large (and small) clusters

- **Idea**: Discover & take advantage of distinctive properties of Big Learning algorithms
  - Use training data to learn parameters of a model
  - **Iterate until Convergence** approach is common
    - E.g., Stochastic Gradient Descent for Matrix Factorization or Multiclass Logistic Regression; LDA via Gibbs Sampling; Page Rank; Deep learning; ...
Parameter Servers for Distributed ML

- Provides all machines with convenient access to global model parameters
- Enables easy conversion of single-machine parallel ML algorithms
  - “Distributed shared memory” programming style
  - Replace local memory access with PS access

```
UpdateVar(i) {
    old = y[i]
    delta = f(old)
    y[i] += delta
}
```

```
UpdateVar(i) {
    old = PS.read(y,i)
    delta = f(old)
    PS.inc(y,i,delta)
}
```

† Ahmed et al. (WSDM’12), Power and Li (OSDI’10)
The Cost of Bulk Synchrony

Wasted computing time!

Threads must wait for each other
End-of-iteration sync gets longer with larger clusters

Precious computing time wasted

But: Fully asynchronous => No algorithm convergence guarantees
Stale Synchronous Parallel (SSP)

Allow threads to *usually* run at own pace
Fastest/slowest threads not allowed to drift >S iterations apart
Protocol: check cache first; if too old, get latest version from network
Consequence: fast threads must check network every iteration
Slow threads check only every S iterations – fewer network accesses, so catch up!

[NIPS’13]
Staleness Sweet Spot

Convergence per iteration

Convergence per second

Iterations per second

Fresher data

Staler data

[ATC'14]
Enhancements to SSP

• Early transmission of larger parameter changes, up to bandwidth limit [SoCC’15]

• Find sets of parameters with weak dependency to compute on in parallel
  – Reduces errors from parallelization

• Low-overhead work migration to eliminate transient straggler effects

• Exploit repeated access patterns of iterative algorithms (IterStore) [SoCC’14]
  – Optimizations: prefetching, parameter data placement, static cache policies, static data structures, NUMA memory management
IterStore: Exploiting Iterativeness

Overall performance: CF, 5 iters

Collaborative Filtering (CF) on NetFlix data set, 8 machines x 64 cores
Big Learning Systems Big Picture

Framework approaches:
- BSP-style approaches: Hadoop, Spark
- Think-like-a-vertex: Pregel, GraphLab
- Parameter server: Yahoo!, SSP

Tend to revisit the same problems
Ad hoc solutions
Unified Scale Down, Scale Up, Scale Out Big Data System?

No system combines all three

Research questions:

- How best to combine: Programming & Performance challenges

- Scale down techniques for Machine Learning?
  E.g., Early iterations on data synopses

- Scale up techniques more broadly applied?
  Lessons from decades of parallel computing research

- Scale out beyond the data center?
  Lessons from IrisNet project?  [Sigmod’03, PC 2003]
How to Tackle the Big Data Performance Challenge

Three approaches to improving performance by orders of magnitude are:

• *Scale down* the amount of data processed or the resources needed to perform the processing

• *Scale up* the computing resources on a node, via parallel processing & faster memory/storage

• *Scale out* the computing to distributed nodes in a cluster/cloud or at the edge

Acknowledgment: Thanks to MANY collaborators
Appendix
References (1/3)

Slides 9-11:


Slides 13-14:


Slide 15:


Slide 24:

References (2/3)

Slide 25:


Slide 27:


Slide 28:


[SPAA’13] see above


References (3/3)


Slide 31:


Slide 33:


Slide 34:


Slides 35-36:


Slide 38:


Acknowledgments

The work presented in this talk resulted from various collaborations with a large number of students, and colleagues. I thank all of my co-authors, whose names appear in the list of References.

A number of these slides were adapted from slides created by my co-authors, and I thank them for those slides.