Big Data, Deep Data, and the Effect of System Architectures on Performance

Peter M. Kogge
McCourtney Prof. of CS & Engr.
University of Notre Dame
IBM Fellow (retired)
This Talk’s Goals

• Start discussion on what really affects Big Data performance

• Focus on high-end “Challenge Problem” that
  – Goes beyond traditional Big Data
  – Requires large-scale parallel systems for solution

• Project performance versus “conventional” systems – both now and in near futures

• Introduce possible alternatives
  – Both in way we express such problems
  – And in way we build systems
Acknowledgements

- Challenge problem and analysis discussed here is outgrowth of extended collaboration with **David Bayliss**, Chief Data Scientist, Lexis Nexis Risk Solutions, Boca Raton, FL

- Pig/ECL results are a subset of a study performed by Keren Ouaknine, Dept. of Computer Science, Hebrew University, Jerusalem, under sponsorship of Lexis Nexis

- The work discussed here was funded in part by the Univ. of Notre Dame, in part by the US Dept. of Energy, Sandia National Labs, as part of their XGC project, and in part by Emu Solutions, Inc.
Topics

• **Scalability and Benchmarking**
• Benchmarking Examples
• The NORA Challenge problem
• Scaling Analysis on Different Systems
• An Alternative
Benchmarking

- Benchmarking: comparing to “industry best”
- Example: Benchmark for purchase of new plane
- 1st Criteria: time from SFO to Beijing
- Is that sufficient?
  - With what passenger/cargo load?
  - With what fuel load?
  - Under what weather conditions?
- Is that sufficient for projecting other trips?
  - Under different payloads
  - New York to SFO?
  - Multi-hop thru Texas?
- Subsidiary metrics often valuable
  - Cruise speed, miles/gallon/ton of cargo, ...
Benchmarking Computers

- Typical 1\textsuperscript{st} Criterion: time to solution
- For computing, need
  - "Benchmark program"
  - "Benchmark Data Set"
  - Reference hardware
- For parallel systems:
  - Architecture/resources per node
  - How many nodes
- What about problem size and execution time?
  - Again as a function of system configuration
- How do we draw conclusions about similar apps?
Today’s Architecture Classes

- **Heavyweight:**
  - Traditional 100+W multi-core
  - High clock rate, big heatsinks
  - Often needs support chip set

- **Lightweight:**
  - Lower power single chip system
  - Lower performance but denser packaging

- **Hybrid:**
  - Heavyweight/GPU combination
  - GPU performs bulk of computation
  - Heavyweight manages data transfers

- **Others later**
Scaling

• Time typically strong function of:
  – # of Nodes
  – Problem Size

• **Weak Scaling**:
  – Time is ~ constant if we
    • Increase # of nodes
    • Increase problem size proportionally
  – Net effect: Constant “data/node”

• **Strong Scaling**:
  – Time decreases if we
    • Increases # of nodes
    • But keep problem size constant

• **Amdahl’s Law**
  – Serial part of parallel program limits speedup

Real Scaling curves seldom this perfect
Topics

- Scalability and Benchmarking
- **Benchmarking Examples**
  - Top500: Computation intensive
  - Graph500: Communication intensive
  - Pig Mix: Map Reduce kernels
- The NORA Challenge problem
- Scaling Analysis on Different Systems
- An Alternative
<table>
<thead>
<tr>
<th><strong>Benchmark</strong></th>
<th><strong>Tested</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>LINPACK</td>
<td>Dense linear algebra</td>
</tr>
<tr>
<td>Breadth First Search</td>
<td>Big graph traversal</td>
</tr>
<tr>
<td>Pig Mix</td>
<td>Hadoop basic queries and scalability</td>
</tr>
<tr>
<td>SWIM</td>
<td>Real life MapReduce workloads</td>
</tr>
<tr>
<td>Berlin SPARQL Benchmarks</td>
<td>e-commerce workload tests</td>
</tr>
<tr>
<td>DBpedia SPARQL Benchmark</td>
<td>query-log mining, clustering, SPARQL feature analysis.</td>
</tr>
<tr>
<td>TPC Benchmarks</td>
<td>SQL transactional and decision support</td>
</tr>
</tbody>
</table>
Supercomputing And LINPACK

- Standardized list of “fastest” 500 supercomputers
  - Lists released twice yearly since 1994
- Benchmark: solving $Ax = b$ for large matrices
- Performance metric: # flops per second
  - $R_{peak}$ = peak capability of machine
  - $R_{max}$ = rate achieved over LINPACK
- Submitter free to select problem size that optimizes

See www.top500.org
Unbroken Growth in Rmax, But Change in Architecture

Did Something Happen in 2004?

1.9X Performance increase per year over 20 years!

Suddenly a Diversity in Architectures

Uniform Architecture

1.E+08
1.E+07
1.E+06
1.E+05
1.E+04
1.E+03
1.E+02
1.E+01

01/01/92
01/01/96
01/01/00
01/01/04
01/01/08
01/01/12
01/01/16

Rmax (Gflops/sec)

Heavyweight (H)  Lightweight (L)  Accelerator (A)  Trend: CAGR=1.88
2004: The Perfect Storm

We hit the wall in terms of cooling chips

- Only remaining option: more cores/chip
- But this complicates programming
- And at the same time we hit limits on communicating off-chip

**Moore’s Law provides more, faster transistors**

After 2004, we cannot use the “faster” part

Which forced clock rates to flatten
Sockets and Cores Growing
Flops/Cycle Skyrocketing: This Drives Programming Complexity
Memory per flop/s is Dropping

- Heavyweight (H)
- Lightweight (L)
- Accelerator (A)

Trend: CAGR=0.76
There Is a Green500 List Also

Overall: increasing energy efficiency lowers performance faster
Graph 500: A “Big Graph” Benchmark

- Newer than TOP500, and communication-oriented
- Start with a vertex in a large graph, find all reachable nodes
- Performance metric: **TEPS**: Traversed Edges/sec

Starting at 1: 1, 0, 3, 2, 9, 5

### Level Scale Size GB Edge(B)

<table>
<thead>
<tr>
<th>Level</th>
<th>Scale</th>
<th>Size</th>
<th>GB</th>
<th>Edge(B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>26</td>
<td>Toy</td>
<td>17</td>
<td>272</td>
</tr>
<tr>
<td>11</td>
<td>29</td>
<td>Mini</td>
<td>140</td>
<td>280</td>
</tr>
<tr>
<td>12</td>
<td>32</td>
<td>Small</td>
<td>1024</td>
<td>256</td>
</tr>
<tr>
<td>13</td>
<td>36</td>
<td>Medium</td>
<td>17408</td>
<td>272</td>
</tr>
<tr>
<td>14</td>
<td>39</td>
<td>Large</td>
<td>143360</td>
<td>280</td>
</tr>
<tr>
<td>15</td>
<td>42</td>
<td>Huge</td>
<td>1153434</td>
<td>281.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Average</td>
<td>273.6</td>
<td></td>
</tr>
</tbody>
</table>

See www.graph500.org
TEPS vs Time

1.0E+05
1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E+00
1.0E-01
1.0E-02

1/1/10  1/1/11  1/2/12  1/1/13  1/1/14

Heavyweight  Lightweight  Hybrid  Other  Trend

62X Increase per Year!
TEPS vs Memory

![Graph showing TEPS (Billions/sec) vs Memory (GB) with data points for Heavyweight, Lightweight, Hybrid, and Other categories.]
TEPS vs # Nodes

The graph shows the relationship between TEPS (Billions/sec) and the number of nodes, represented by "Nodes". The trend line indicates a power law relationship with a coefficient of 0.92, suggesting that as the number of nodes increases, the TEPS also increases at a rate described by this coefficient. The legend indicates different categories of nodes: Heavyweight, Lightweight, Hybrid, Other, and Trend.
Where Does Improvement Come From? BlueGene Analysis

- **Software/Algorithm Improvements**
- **Due to Hardware**

![Graph showing GTEPS per Node]

- BlueGene P
- BlueGene Q
Sample Big Data Scaling Study: the Ouaknine Study

• Understand sensitivity to problem size of 17 PIG MIX benchmarks
  – With input sets from 2 million to 100 million rows

• Compare three implementation paths:
  – Pig Latin baseline
  – Direct Hadoop Implementation
  – PIG translated via software tool BACON to Lexis Nexis ECL

• Use Common Hardware Platform
  – 401 node Thor cluster
  – 300GB, 10K RPM disks
Baseline PIG Execution Times

- Times largely linear with size after ~20M rows
- Sort-heavy benchmarks by far the most time-consuming
Speedup: Pure MapReduce vs Baseline

On average 13% MapReduce Advantage for large data sets
Speedup: BACON ECL vs Baseline

Huge ECL advantage
For smaller data sets

Order by a single value

Wide Key Group

Almost 2X for large data sets – and its all in the compiler!
Topics

• Scalability and Benchmarking
• Benchmarking Examples
• The NORA Challenge problem
• Scaling Analysis on Different Systems
• An Alternative
LexisNexis Challenge Problems

• **CP1:** MASSIVE Intermediate data set
  - Huge self join
  - Disk & network bound

• **CP2:** MASSIVE computation problem
  - Avoid even bigger join by
    - 100s of “smaller” joins,
    - And very expensive “scoring function”

• **CP3:** Real-time response problem
  - Use results of batch runs of above as indexable sets

• **CP4:** Ultimate version of CP1 & 2.
  - Join condition is “OR” of hundreds of terms,
  - With very complex scoring function on each result
  - And sophisticated filter to reduce results

Today: batch mode once/week @ hours per run
An Example Challenge Problem #1

Today

- 40+ TB of Raw Data
- Periodically clean up & combine to 4-7 TB
- Weekly “Boil the Ocean” to precompute answers to all standard queries
  - Does W have financial difficulties?
  - Does X have legal problems?
  - Has Y had significant driving problems?
  - Who has shared addresses with Z?
  - ... 

Auto Insurance Co: “Tell me about giving auto policy to Jane Doe” in < 0.1 sec

Look up answers to precomputed queries for “Jane Doe” and combine

“Jane Doe has no indicators
But she has shared multiple addresses with Joe Scofflaw
Who has the following negative indicators ....”
A Key Query from this Problem

- Given: 14.2 billion records from
  - 800 million *entities* (North American people, businesses)
  - 100 million *addresses*

- Goal: given entity ID, find all other IDs such that
  - Share at least 2 addresses in common
  - Or have one address in common and last name that is “close”

- Matching last names requires processing to check for typos (“Levenshtein distance”)
Challenge #1: Runaway Intermediate Data

14.2B recs
325 B/rec
4.6TB

800M distinct IDs

14.2B recs
100+ B/rec
1.5TB

Project

Join on Address

300+TB

1.6T recs
200+ B/rec

• Compute adr hash
• Compare lnames
• Init score to 3
• Project

Sort & Remove Duplicates

1.5T recs
30B/rec
45TB

Group by ID pairs & Sum scores, Lname_match

1.2T recs
16B/rec
20TB

Select on Score & Lname_match

12B recs
16B/rec
200GB

400M distinct IDs

Hash ID1,2 & Distribute

Send between nodes via TCP/IP datagrams

12B recs
16B/rec
200GB

“h”

“t”

“D”

“J”

“D”

48+TB

{(ID1, ID2, adrhash, score, lname_match)}

{(ID, lname, adr)}
Pipelined Parallel Algorithm: (Assume 400 Parallel Nodes)

1. Read in slice of local partition
2. Project out (ID, Iname, address fields)
3. Use hash of address to bin into 400 buckets
4. Distribute (ID, Iname, adr) to hashed node
5. Use large hash table to arrange for each address a set of (ID, Iname)
6. Generate all non-redundant pairs from each set, including address edit distance, and distribute on basis of hash of ID pairs
7. Use 2D hash table to bin by ID pairs, and dump to local disk by bin
8. For each bin use large hash table to match up pairs and perform aggregation
9. Perform filter checks and output

184 Iterations
64MB per slice

250K Iterations
1/Unique Address

Barrier

Barrier
Topics

- Scalability and Benchmarking
- Benchmarking Examples
- The NORA Challenge problem
- **Scaling Analysis on Different Systems**
- An Alternative
Heavyweight Configurations

• Baseline: Lexis Nexis HPCC Configuration
  – 100 4-node Blades in 10 racks
  – Each node:
    • 1 effective disk spindle: 2TB, 0.16 GB/s sustained
    • 2 sockets with 6 cores each @ 2.4GHz
    • 96 GB of DDR3 on 3 ports
      – With 10.66GB/s peak transfer rate
    • 1 network port @ 0.1 GB/s (1 Gbps Ethernet)
  – Star topology with external routers

• Memory Rich Configuration
  – Same as above but with maxed DRAM for RAM Disk

• 2015 Configuration
  – 4X cores/socket, DRAM, switched Infiniband

• 2015 Configuration with DRAM for RAM Disk
Possible “Lightweight” System

- Assume Calxeda System on a Chip
  - 4 1-1.4GHz ARM A9 cores w’FPU
  - Single DDR3 2 rank controller
  - Networking: GigE, XAU
  - Supports up to 5 SATA
  - Fabric: 8x8 crossbar, 10Gbps links
    - 3 internal, 5 external

- Calxeda Reference card:
  - 4 SOCs + 4 VLP DDR3 DIMMs (max 4GB each)
  - 4 SATA sockets/SOC for disk connections
  - 8 interfaces for off-card fabric

- 2U Blade (based on Boston Viridis Chassis)
  - 12 reference cards + up to 24 SATA

- Assumed Configuration of 40 blades, 2 racks
Emerging Technology: 3D Stacks

- Stackable memory chips
- “Through Silicon Vias” (TSVs)
- Logic chip on bottom
  - Multiple memory controllers
  - More sophisticated off-stack interfaces than DDR
- Prototype demonstrated in 2011
- 1st Product expected in 2015 timeframe
  - Spec: http://www.hybridmemorycube.org
  - Capacity: up to 8GB: 8X single chip
  - Bandwidth: up to 480GB/s: 40X
  - Lots of room on logic chip
- Bottom Line: Huge increase in
  - Memory density
  - Bandwidth

http://www.micron.com/products/hybrid-memory-cube
Possible X-Caliber Architecture (2016+)

Each Stack
- 32 GB DRAM
- 256GB PCM
- Logic chip at bottom
- 64 0.5GB “Vaults”
- 8 full-duplex links – 32 GB/s each dir

X-caliber Node Mockup

M’s built from 3D stacks of memory

Memory System (M) and Embedded Memory Processor (EMP)
- Two computation Units
  - Right next to the DRAM vault memory controller (VAU)
  - To aggregate between DRAM vaults (EMP)
- “Memory Network” Centric
- Homenode for all addresses
  - Owns the address, data, and its state, “coherency”
- Three Control-Flow Options
  - In the Processor (“Memory is the Accelerator”), conventional
  - In the Memory System (“Processor is the Accelerator”), our approach
  - Both, probably un-programmable
- At 1-2 GHz, 4 EMPs per vault
- 64 vaults
- 2-4K threads per node in the memory system!
Thought Experiment: Memory Stack Only Version

- Same stack as from X-caliber
  - Multiple DRAM, NVRAM vaults
  - Internal crossbar for full interconnect
  - 8 external ports
- Multiple stacks on something like a DIMM
- Remove Processor sockets and NIC chips
- Use stack external ports for all routing
- Keep routing on global address
- And grow up logic chip processing
Modeling

• For each of 9 steps compute usage of resources:
  – Disk bandwidth
  – Instructions per second
  – Memory Bandwidth
  – Network bandwidth

• All as function of data set size and partition size

• As appropriate, pipeline and overlap
  – Account for overlapping use of resources
  – Time per iteration: maximally used resource

• Deliberate simplifications
  – No OS overheads included
  – No statistical variations assumed
Today’s Algorithm on Baseline

(a) Wall clock time (in sec) by step

(b) Resource utilization by step

(c) Resource utilization vs. Wall Clock

(d) Breakdown of overall resources

Note Logarithmic Y Axis
## Details: Heavyweight Alternatives

### Baseline: 1026s

<table>
<thead>
<tr>
<th>Step #</th>
<th>Disk</th>
<th>CPU</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Baseline with RAM disk: 708s

<table>
<thead>
<tr>
<th>Step #</th>
<th>Disk</th>
<th>CPU</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2015 Heavyweight: 1.59

<table>
<thead>
<tr>
<th>Step #</th>
<th>Disk</th>
<th>CPU</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2015 Heavyweight with RAM Disk: 8.12

<table>
<thead>
<tr>
<th>Step #</th>
<th>Disk</th>
<th>CPU</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Heavyweight Resource Utilization

Baseline

Baseline with RAM disk

2015 Heavyweight

2015 Heavyweight with RAM Disk
Non-Heavyweights

Baseline: 1026s

Resources Used/node (sec)

Step #

Disk
CPU
Memory
Network

Lightweight: 784s

Resources Used/node (sec)

Step #

Disk
CPU
Memory
Network

Xcaliber: 86s

Resources Used/node (sec)

Step #

Disk
CPU
Memory
Network

Xcaliber Stacks Only: 67s

Resources Used/node (sec)

Step #

Disk
CPU
Memory
Network
Non-Heavyweight Utilization

Baseline

Lightweight

Xcaliber

Stacks Only
Comparison

- **3D Stack only:** 67X speedup in 1/10th the hardware
- **XCaliber**

<table>
<thead>
<tr>
<th>Size in Racks</th>
<th>Speedup over Baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

- **Heavyweight**
- **Lightweight**
- **3D**

2015 with RAMDisk
2015 RAMDisk Baseline

BDDAC Keynote: May 21, 2013
“Balancing” the Design

- Reduce under-utilized resources until ~ balanced but still close to prior execution time

Balancing Baseline Parameters
Time = 1090 sec = 6% increase

Balancing Stack-only Parameters
Time = 23 sec = Still 44X

%# under resource name are amount after reduction relative to start
Topics

• Scalability and Benchmarking
• Benchmarking Examples
• The NORA Challenge problem
• Comparative Analysis
• An Alternative
Why Look for an Alternative

- Huge computational effort to “boil the ocean”
  - Most of which is never used
- Slow response to new data (wait for next boil)
- No help for queries that aren’t quite expressible from output of boil
• **Create graph** of name/address records

  ■ Entity ID

  ★ Last Name

  □ Address

• **Query:** Given a specific ID1 find all ID2s that meet requirements
  - Start with ID1 and find all ID2 reachable via shared address
  - Score each path
  - Sum all path scores & pass (ID1, ID2) if > threshold
An Example as Seen In Memory

Query: Given a particular ID, output all other IDs that pass test

IDs Last Names

Addresses

96GB

18GB

300GB
A Query

Start with ID, follow last names to all matching address, and then other IDs
What Is Needed For Fast Implementation?

- Very Large Shared Memory space
  - 400+GB needed for *just* this one graph
  - Adding more attributes requires more 300GB “sparse matrices”
  - Easily expand to TBs of memory
- Fast “pointer following”
- Fast spawning of additional threads to follow alternatives
- Atomic pointer updates to “add” new vertices

**Observation:** Conventional architectures offer poor support
- Large amounts of memory require multiple “server nodes”
- With highly expensive software stacks for remote memory access
- Almost no memory reuse – caches worthless
- Software multi-threading is expensive
Massive Multi-Threaded

- One node:
  - 2\textsuperscript{nd} gen XMT-2 500MHz 128 thread Threadstorm chip
  - Very cheap thread creation
- Up to 8192 node systems
- Up to 512TB total
- “Shared memory” between nodes

http://www.adms-conf.org/uRiKA_ADMS_keynote.pdf
Traveling Threadlets

- Single Address Space Visible to all Hosts & Gossamer Cores
- Hosts can launch:
  - Reads and Writes of Memory
  - Threadlets for execution on Gossamer core
- Gossamer Cores can
  - Spawn new threadlets
  - Migrate threadlets to other cores
- Memory need not be just today’s DRAM

Performance Projections

Response Time/Query:

- 10,000+ microseconds for lightweight, multi-threaded
- ~40 microseconds for threadlet
Overall Summary

- Big Data: lacking the performance insight of “mature” traditional applications
- Performance understanding is more than execution time
- Understanding “tall poles” requires lower level metrics
- NORA: outstandingly rich Big Data problem
- Graph-based approach: more “real-time”
- Today’s Architectures: at best limited improvement
- Emerging architectures have potential to be “business model changing”