Leveraging HPC Expertise and Technology in Data Analytics

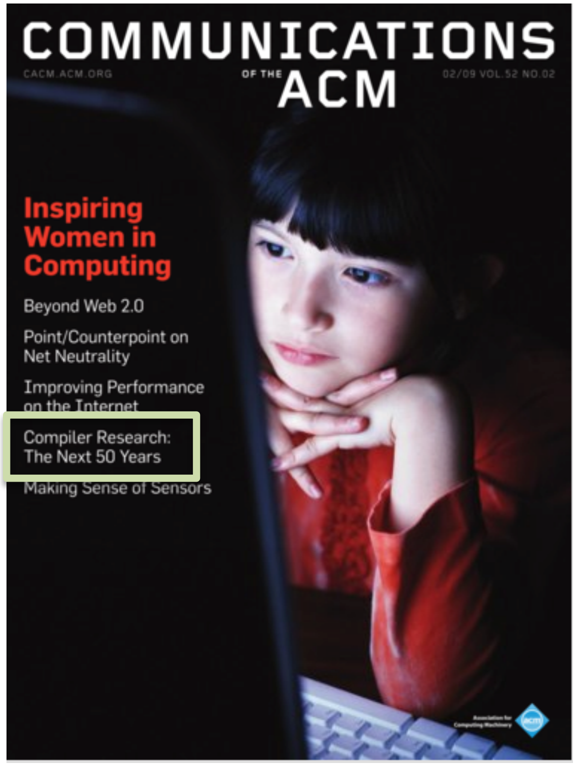
Mary Hall

September, 2014

References for this talk

- **Rethinking Abstractions for Big Data: Why, Where, How and What**, M. Hall, R.M. Kirby, F. Li, M. Meyer, V. Pascucci, J. Phillips, R. Ricci, J. van der Merwe, S. Venkatasubramanian, arXiv, June 2013.
- **Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets,** N. Satish, N. Sundaram, M.A. Patwary, J. Seo, J. Park, M.A. Hassaan, S. Sengupta, Z. Yin, P. Dubey, SIGMOD 2014.
- **Spark: Cluster Computing with Working Sets,** M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, HotCloud 2010.

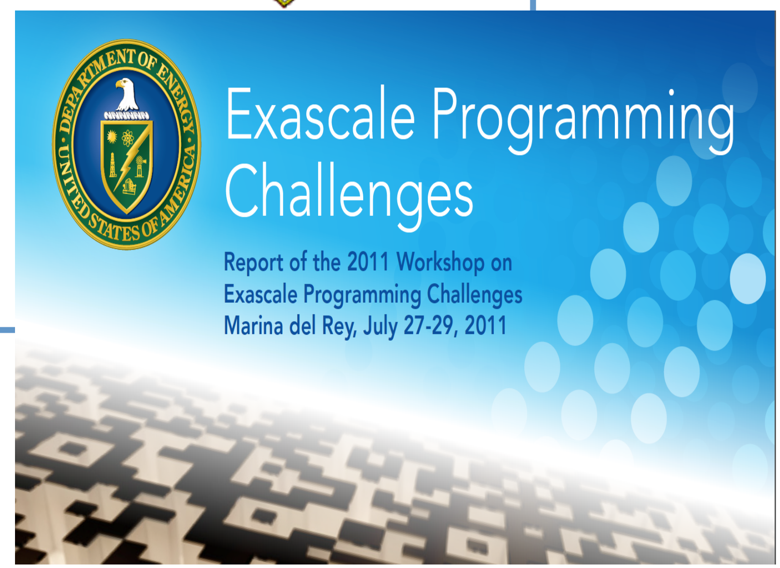
Future of the Field: Programming Technology and Compilers



ExaScale Software Study: Software Challenges in Extreme Scale Systems

Saman Amarasinghe
Dan Campbell
William Carlson
Andrew Chien
William Dally
Elmootazbellah Elnohazy
Mary Hall
Robert Harrison
William Harrod
Kerry Hill
Jon Hiller
Sherman Karp
Charles Koelbel
David Koester
Peter Kogge
John Levesque
Daniel Reed
Vivek Sarkar, Editor & Study Lead
Robert Schreiber
Mark Richards
Al Scarpelli
John Shalf
Allan Snavely
Thomas Sterling

September 14, 2009



Autotuning!

Programming Systems Targeting Exascale

Thanks to exascale reports and workshops

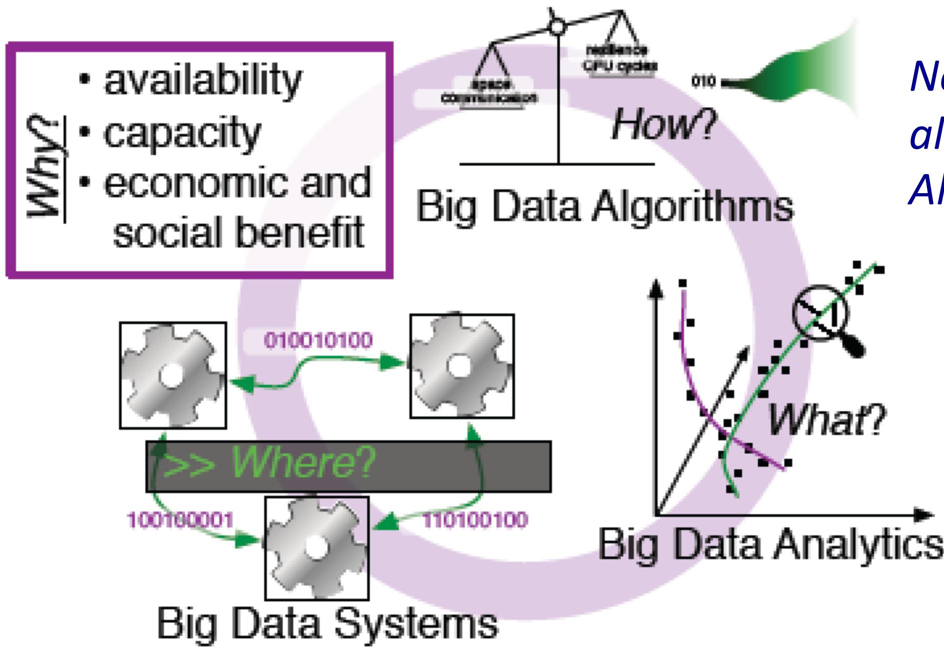
- Multiresolution programming systems for different users
  - Joe/Stephanie/Doug **[Pingali, UT]**
  - Elvis/Mort/Einstein **[Intel]**
- Specialization simplifies and improves efficiency
  - Target specific user needs with domain-specific languages/libraries
  - Customize libraries for application needs and execution context
- Interface to programmers and runtime/hardware
  - Seamless integration of compiler with programmer guidance and dynamic feedback from runtime
- Toolkits rather than monolithic systems
  - Layers support different user capability
  - Collaborative ecosystem
- Virtualization (over-decomposition)
  - Hierarchical, or flat but construct hierarchy when applicable?

THE UNIVERSITY OF UTAH

Big Data vs. HPC: Fundamentally Different?

	Big Data	HPC
Applications	Data analytics: Social networks, industry	Large-scale scientific simulation: government, industry
Characterized by	Typically, independent file operations, database queries	Typically map to 3-D grid to represent physical space
Prevalent data abstractions	Graphs (sparse), databases, text files	Arrays (dense and sparse), objects
Programming Models	Map-Reduce/HIVE/Giraph etc.	MPI/OpenMP/CUDA widely used
Failure Model	Assume failures common, need to be tolerated	Assume failures infrequent (spend $)
System Cost	Use the technology with the best price-performance ratio	Use the fastest possible processors/network

Rethinking Abstractions for Big Data



Need for appropriate models for algorithm development.
Algorithmically managing inaccuracies.

Reducing dimensionality, not drawing false conclusions.
Composing data models.
Visualizing for user involvement (overview+detail).

Similar looming systems and programming issues in HPC and Big Data (productivity, resilience, energy efficiency).

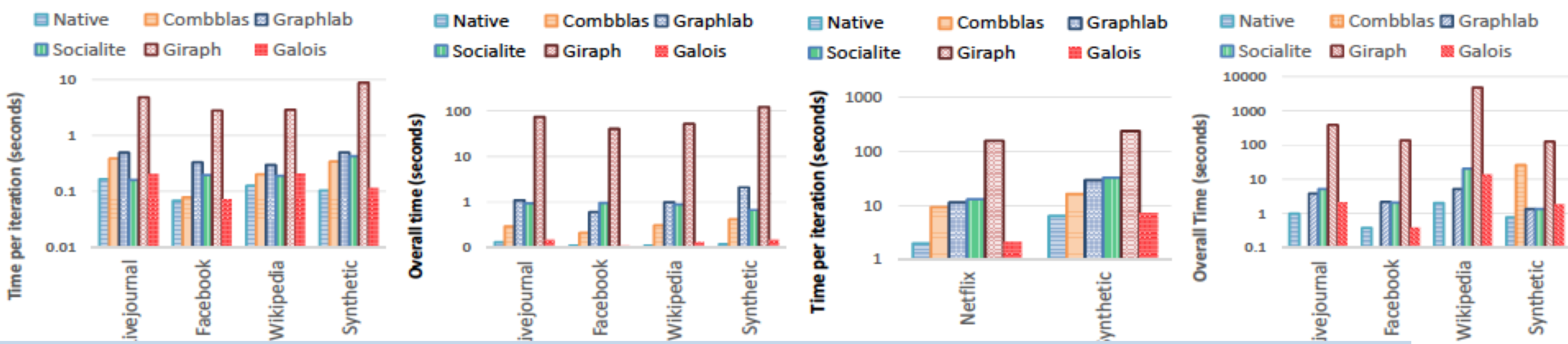
Example 1: Graph Algorithms

- Performance issues in algorithms on sparse graphs
  - Overhead of partitioning
  - Load imbalance
  - Small computation relative to memory access and communication (high memory-to-compute ratio)
- Recent study: Compares 5 different graph frameworks to hand-coded graph algorithms
  - **Parallel computing:** GraphLab, Galois
  - **Distributed memory:** CombBLAS
  - **Big data:** Giraph, SociaLite

THE UNIVERSITY OF UTAH

Graph Algorithms, Performance

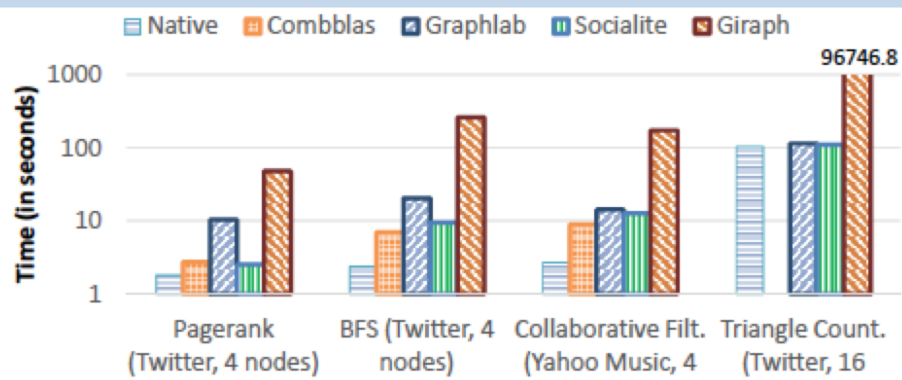
Single socket performance, Intel Xeon E5-2697 (24 cores)



counting

Conclusions: Thread-level parallelism is essential, data representations impact performance, Hadoop-based framework has lowest performance

Multiple sockets, larger graphs (Infiniband interconnect)



Conclusions: MPI performs better than Linux sockets, low memory footprint improves performance, graph partitioning important, triangle counting dominated by network traffic, Hadoop-based framework has lowest performance

THE UNIVERSITY OF UTAH

Programming, from an HPC Perspective

- Disadvantages of Map-Reduce
  - Map/Reduce not always the right abstraction
  - Lots of new frameworks resulted: HIVE, BigTable, Dremel, Giraph, …, Cloud Dataflow
- Fundamental changes needed
  - I/O-based programming models are inefficient
  - Efficiency comes from customization and specialization (low level?)
  - On today's architectures, fully exploiting locality and parallelism is essential (low level?)

Example 2: SPARK (Berkeley)

Goal: Support for computations that cannot be expressed as "asynchronous data flows"

- Iterative computations
- Interactive analytics

Key abstraction: *resilient distributed data set (RDD)*

- A read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost (in-memory plus fault tolerance)
- Users can explicitly cache an RDD in memory across machines and reuse it in multiple parallel operations

* Integrated into Cloudera, the largest distribution of Hadoop

Some concepts

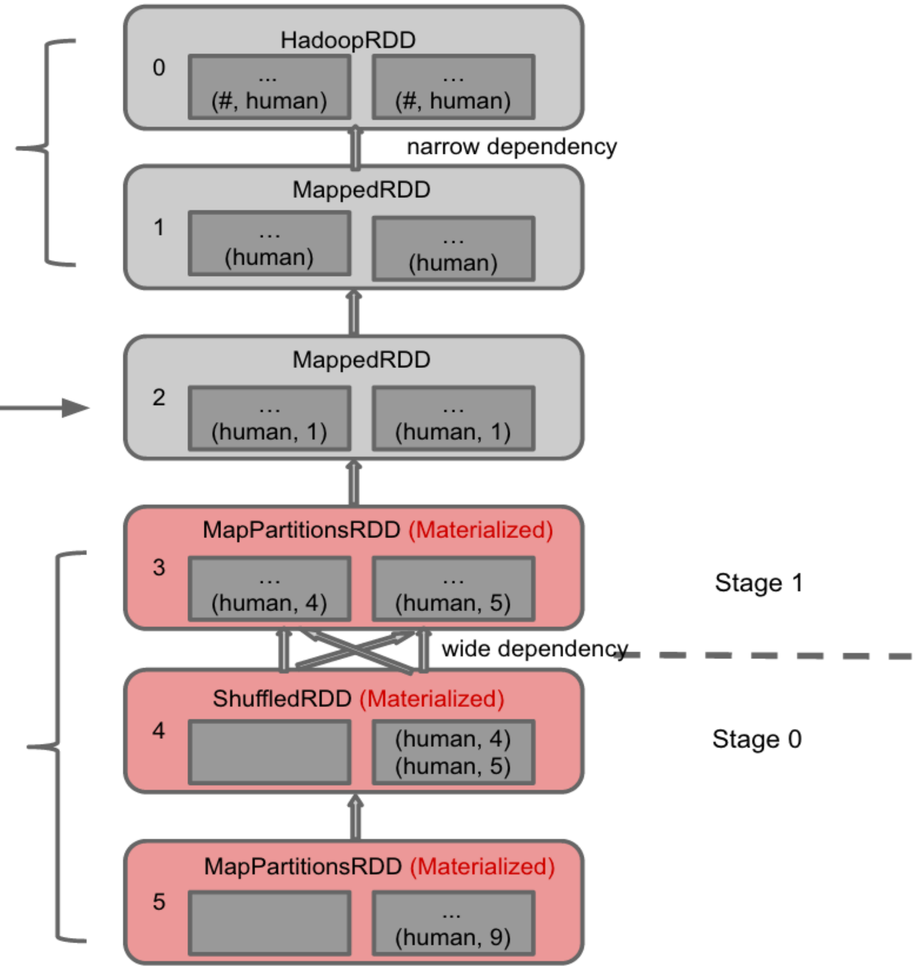
- Dependences govern generation of stages and RDDs.
- Users designate RDDs as cached (one in PageRank example)
- Materialization points (boxes colored red) generate the RDDs (o/w they are lazy)
- Shuffles are global synchronization points, require significant data movement

Word Count in SPARK

```
val file  = sc.textFile("README.md")

val count = file.map(word => (word,1))
                .reduceByKey(_+_)

count.toArray
```



PageRank in SPARK

Calculation:

$$PR^{t+1}(i) = r + (1-r) * \sum_{j|(j,i)\in E} \frac{PR^t(j)}{\text{degree}(j)}$$

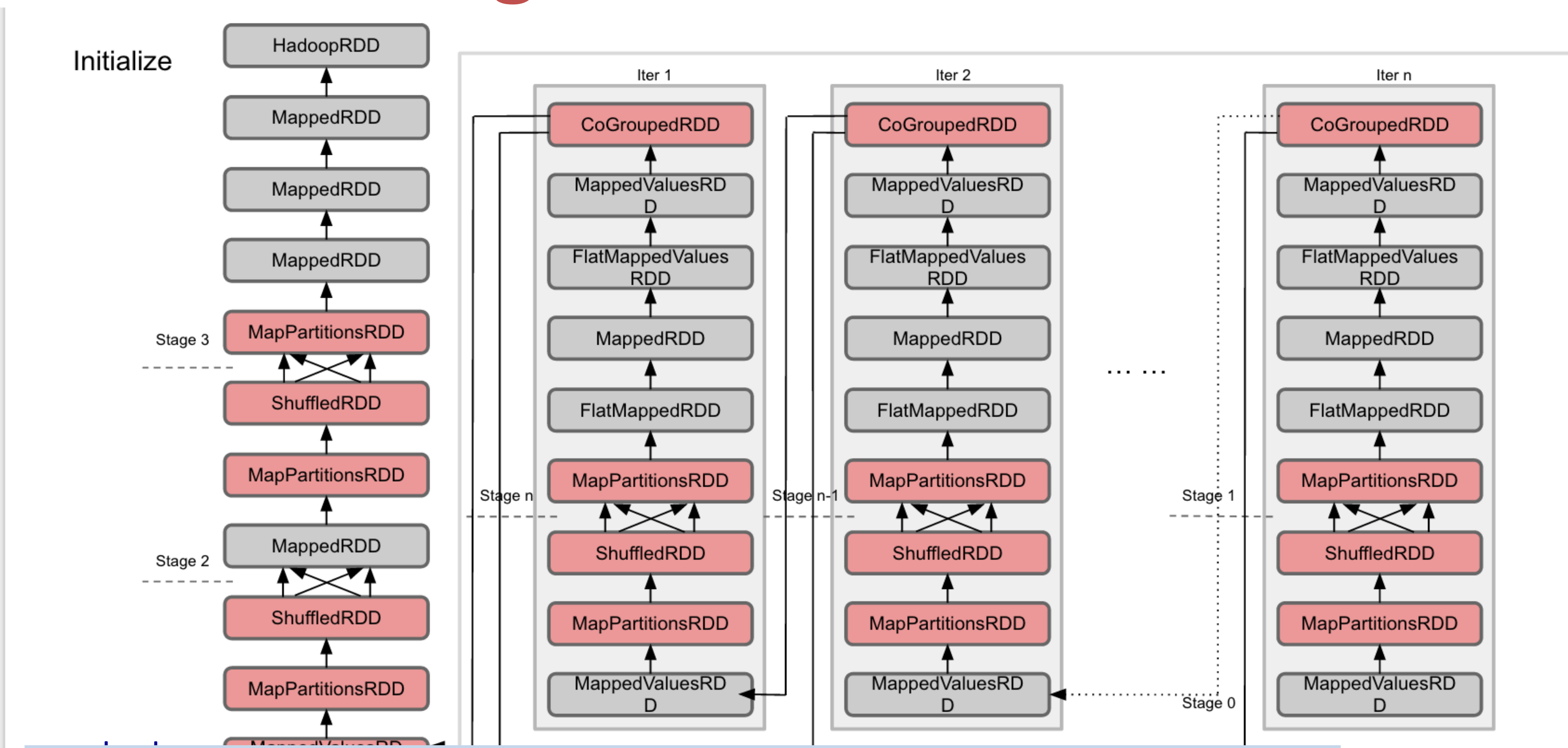
```
val lines = ctx.textFile(args(0), 2)
 val links = lines.map{ s =>
  val parts = s.split("\\s+")
  (parts(0), parts(1))
 }.distinct().groupByKey().cache()
 var ranks = links.mapValues(v => 1.0)

 for (i <- 1 to iters) {
  val contribs = links.join(ranks).values.flatMap{ case (urls, rank) =>
  val size = urls.size
  urls.map(url => (url, rank / size))
 }
 ranks = contribs.reduceByKey(_ + _).mapValues(0.15 + 0.85 * _)
 }

 val output = ranks.collect()
 output.foreach(tup => println(tup._1 + " has rank: " + tup._2 + "."))
```


THE UNIVERSITY OF UTAH

PageRank in SPARK



Conclusions: Many dependences represent dataflows or pipelines (simplification?), global synchronization at each iteration (optimize?), opportunities for exploiting fine-grained parallelism

THE UNIVERSITY OF UTAH

Summary: Opportunities and Challenges

- Managing data movement critical to performance in both communities
  - Exploit data reuse, optimize communication, avoid unnecessary file I/O
- Exploiting architecture features will dramatically improve performance
  - Parallelism at all levels (ILP, SIMD, threads, processes)
  - Data affinity
- Multiresolution programming systems will be useful to both communities
- Similar issues are looming: energy and resilience

THE UNIVERSITY OF UTAH